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; Fiterău-Broştean 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; Rodríguez 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; Rodríguez 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, `tid`, 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 `tid=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 to 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.

```
tasks1 = ⟦get tasks whose id=tid⟧
print(tasks1.title)

if tasks1:
  comments = ⟦get comments whose task_id=tid⟧
  print(comments.content)

  for c in comments:
    cid = c.commenter_id
    users1 = ⟦get users whose id=cid⟧
    print(users1.name)
    tasks2 = ⟦get tasks whose creator_id=cid⟧
    print(tasks2.title)

  aid = tasks1.assignee_id
  users2 = ⟦get useres whose id=aid⟧
  print(users2.name)
  tasks3 = ⟦get tasks whose creator_id=aid⟧
  print(tasks3.title)
```

Fig. 1. Example program pseudo code

(a) Table tasks

id	title	creator_id	assignee_id
1	1	4	6
2	5	4	6

(b) Table comments

id	task_id	commenter_id	content
3	2	4	7
5	2	6	7

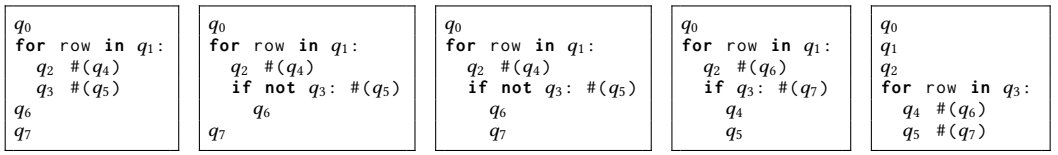
```
q0 : SELECT * FROM tasks WHERE id = 2
q1 : SELECT * FROM comments WHERE task_id = 2
q2 : SELECT * FROM users WHERE id = 4
q3 : SELECT * FROM tasks WHERE creator_id = 4
q4 : SELECT * FROM users WHERE id = 6
q5 : SELECT * FROM tasks WHERE creator_id = 6
q6 : SELECT * FROM users WHERE id = 6
q7 : SELECT * FROM tasks WHERE creator_id = 6
```

(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

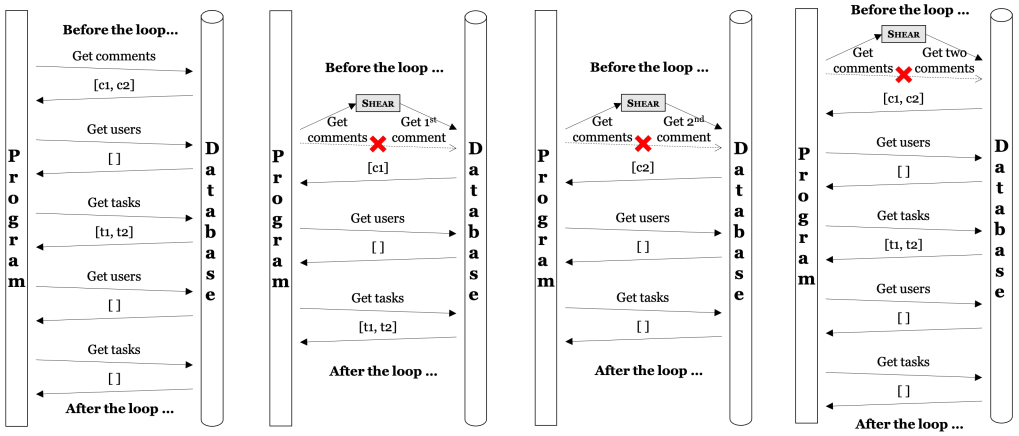
Shear uses this number to identify queries that would be produced by hypothetical loop iterations. Specifically, Shear removes from each altered trace the last 2 queries and the queries before the hypothetical loop. Remaining queries in the first altered trace are q_2 and q_3, which would be produced 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



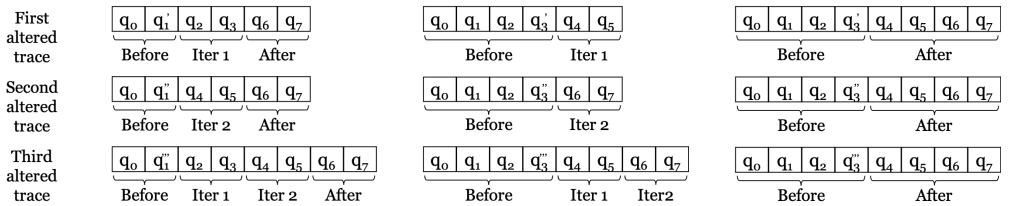
(a) Correct Plausible Loop L (b) Incorrect Plausible Loop W (c) Incorrect Plausible Loop X (d) Incorrect Plausible Loop Y (e) Incorrect Plausible Loop Z

Fig. 3. Plausible loop structures that may produce the example trace



(a) Intact trace (b) First altered trace (c) Second altered trace (d) Third altered trace

Fig. 4. SHEAR alters the database traffic during program execution to detect loops in the example trace



(a) Traces that confirm L (b) Expected if Z were correct (c) Traces that reject Z

Fig. 5. SHEAR compares three altered traces to determine if a hypothetical loop structure is correct

that the program behavior is consistent with the existence of a hypothetical loop. SHEAR therefore determines that a loop indeed iterates over the two rows retrieved by query q_1.

Detecting Non-Loop At Query q_3: Because query q_3 also retrieved two rows during execution, there can potentially be a loop that iterates over the two rows retrieved by query q_3 (Plausible Loop Z). To determine whether this loop exists, SHEAR alters the database traffic for query q_3 to obtain three altered traces. The first altered execution alters query q_3 into query q_3' which retrieves only the first row in query q_3. The resulting altered trace consists of queries q_0, q_1, q_2, q_3', q_4, q_5, q_6, and q_7. The second altered execution alters query q_3 into query q_3'' which retrieves only the second row in query q_3. The resulting altered trace consists of queries q_0, q_1, q_2, q_3'', q_4, q_5, q_6, and q_7. The third altered execution alters query q_3 into query q_3''' which retrieves both the first and the second rows in query q_3. The resulting altered trace consists of queries q_0, q_1, q_2, q_3''', q_4, q_5, q_6, and q_7.

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 no comprise a strict prefix of the queries produced by both of 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.

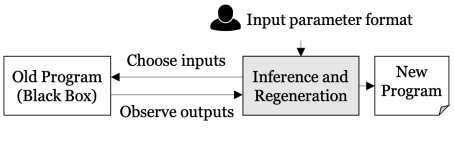
Discussion: Active loop detection is based on several manipulated executions of the program per hypothetical loop. The algorithm works precisely with programs that may contain a variety of loop and repetitive structures including nested loops, consecutive loops, loops with conditional statements, and 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

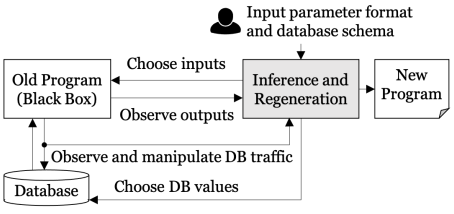
SHEAR instantiates active loop detection on a prior technique called Konure that infers and regenerates database applications [Shen and Rinard 2019]. We reiterate several key concepts from Konure and generalize them to express the new loop and repetitive structures supported by SHEAR.

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.



(a) Generic concept



(b) Specialized for database-backed programs

Fig. 6. Program inference and regeneration

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.

Prog	:=	ϵ | Seq | If | For
Seq	:=	Query Prog
If	:=	`if` Query `then` Prog `else` Prog
For	:=	`for` Query `do` Prog ; Prog
Query	:=	$y \leftarrow$ `select` Col$^+$ `where` Expr `print` Orig*
Expr	:=	`true` | Expr $\wedge$ Expr | Col = Col | Col = Orig
Col	:=	$t.c$
Orig	:=	x | y.Col

$x, y \in \mathit{Variable}, \quad t \in \mathit{Table}, \quad c \in \mathit{Column}$

Fig. 7. Grammar for the Shear DSL

$$\sigma \in \mathit{Context} = \mathit{Input} \times \mathit{Database} \times \mathit{Result}$$
$$\sigma_I \in \mathit{Input} = \mathit{Variable} \to \mathit{Value}$$
$$\sigma_D \in \mathit{Database} = \mathit{Table} \to \mathbb{Z}_{>0} \to \mathit{Column} \to \mathit{Value}$$
$$\sigma_R \in \mathit{Result} = \mathit{Variable} \to \mathbb{Z}_{>0} \to \mathit{Table} \to \mathit{Column} \to \mathit{Value}$$
$$\mathit{Value} = \mathit{Int} \cup \mathit{String}$$

Fig. 8. Contexts for programs in the DSL

The SHEAR DSL is the set of programs $\mathcal{S} \subset$ 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 $\sigma = \langle \sigma_I, \sigma_D, \sigma_R \rangle \in$ *Context* (Figure 8) contains value mappings for the input parameters ($\sigma_I \in$ *Input*), database contents ($\sigma_D \in$ *Database*), and results retrieved by database queries ($\sigma_R \in$ *Result*).

$$\mathrm{T} \quad ::= \quad \mathrm{Nil} \mid (Q, r); \mathrm{T} \mid (Q, r)[\mathrm{T}_1 \ldots \mathrm{T}_r]; \mathrm{T}_0$$
$$Q \in \mathrm{Query}, \quad r \in \mathbb{Z}_{\geq 0}$$

Fig. 9. Grammar for loop layout trees

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

Definition 3. A *query-result pair* (Q, r) has a query $Q \in$ 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 $P \in$ 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 $(\mathrm{T}_1, \ldots, \mathrm{T}_r)$ 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 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 $Q \in$ 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) λ is a non-negative integer that indicates the iteration index or (2) $\lambda =$ AfterLoop which indicates execution of the subprogram that follows the loop. If no such loop was found then $\lambda =$ NotLoop. Each path from the root of the loop layout tree to a leaf generates a corresponding annotated trace.

Definition 6. A *path constraint* $W = (\langle Q_1, r_1, d_1, a_1 \rangle, \ldots, \langle Q_n, r_n, d_n, a_n \rangle)$ consists of a sequence of queries $Q_1, \ldots, Q_n \in$ 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 $(= 0)$, (≥ 1), (≥ 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.

$$\frac{}{\sigma \vdash \epsilon \Downarrow_{\text{exec}} \text{Nil}} \quad \text{(epsilon)}$$

$$\frac{\sigma[Q.y \mapsto \sigma(Q)] \vdash P \Downarrow_{\text{exec}} e}{\sigma \vdash Q\ P \Downarrow_{\text{exec}} (Q, |\sigma(Q)|) @ e} \quad \text{(seq)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma \vdash P_2 \Downarrow_{\text{exec}} e}{\sigma \vdash \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \Downarrow_{\text{exec}} (Q, 0) @ e} \quad \text{(if-0)}$$

$$\frac{|\sigma(Q)| > 0 \qquad \sigma[Q.y \mapsto \sigma(Q)] \vdash P_1 \Downarrow_{\text{exec}} e}{\sigma \vdash \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \Downarrow_{\text{exec}} (Q, |\sigma(Q)|) @ e} \quad \text{(if-1)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma \vdash P_2 \Downarrow_{\text{exec}} e}{\sigma \vdash \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2 \Downarrow_{\text{exec}} (Q, 0) @ e} \quad \text{(for-0)}$$

$$\frac{\sigma(Q) = (x_1, \ldots, x_r) \qquad r > 0 \qquad \sigma \vdash P_2 \Downarrow_{\text{exec}} e \qquad \sigma[Q.y \mapsto x_i] \vdash P_1 \Downarrow_{\text{exec}} e_i \quad \text{for all } i = 1, \ldots, r}{\sigma \vdash \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2 \Downarrow_{\text{exec}} (Q, r) @ e_1 @ \ldots @ e_r @ e} \quad \text{(for-r)}$$

$P, P_1, P_2 \in \text{Prog}, \quad Q \in \text{Query}, \quad \sigma \in \textit{Context}, \quad y \in \textit{Variable},$
$r, i \in \mathbb{Z}_{\geq 0}, \quad x_1, \ldots, x_r \in \text{CRow}$

Fig. 10. Semantics for executing a program using a context to directly obtain a list of query-result pairs

Definition 8. For a program $P \in \text{Prog}$ and a context $\sigma \in \textit{Context}$, $\sigma \vdash P \Downarrow_{\text{exec}} e$ and $\sigma \vdash P \Downarrow_{\text{loops}} l$ denote evaluating P in σ 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 $\diamond$. For any program $P \in \text{Prog}$, $\widetilde{P}$ 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 $\mathcal{S} \subset \text{Prog}$ defined as:

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

The predicate $\mathcal{D}(\widetilde{P}) = \texttt{true}$ states that the two branches of any If statement in $\widetilde{P}$ 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

$$\frac{}{\sigma \vdash \epsilon \Downarrow_{\text{loops}} \text{Nil}} \quad \text{(epsilon)}$$

$$\frac{\sigma[Q.y \mapsto \sigma(Q)] \vdash P \Downarrow_{\text{loops}} l}{\sigma \vdash Q\ P \Downarrow_{\text{loops}} (Q, |\sigma(Q)|) ; l} \quad \text{(seq)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma \vdash P_2 \Downarrow_{\text{loops}} l}{\sigma \vdash \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \Downarrow_{\text{loops}} (Q, 0) ; l} \quad \text{(if-0)}$$

$$\frac{|\sigma(Q)| > 0 \qquad \sigma[Q.y \mapsto \sigma(Q)] \vdash P_1 \Downarrow_{\text{loops}} l}{\sigma \vdash \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2 \Downarrow_{\text{loops}} (Q, |\sigma(Q)|) ; l} \quad \text{(if-1)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma \vdash P_2 \Downarrow_{\text{loops}} l}{\sigma \vdash \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2 \Downarrow_{\text{loops}} (Q, 0) ; l} \quad \text{(for-0)}$$

$$\frac{|\sigma(Q)| = 1 \qquad \sigma[Q.y \mapsto \sigma(Q)] \vdash P_1 \Downarrow_{\text{loops}} l_1 \qquad \sigma \vdash P_2 \Downarrow_{\text{loops}} l_2}{\sigma \vdash \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2 \Downarrow_{\text{loops}} (Q, 1) ; \textsc{ConnectTrees}(l_1, l_2)} \quad \text{(for-1)}$$

$$\frac{\sigma(Q) = (x_1, \ldots, x_r) \qquad r \geq 2 \qquad \sigma \vdash P_2 \Downarrow_{\text{loops}} l \qquad \sigma[Q.y \mapsto x_i] \vdash P_1 \Downarrow_{\text{loops}} l_i \quad \text{for all } i = 1, \ldots, r}{\sigma \vdash \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2 \Downarrow_{\text{loops}} (Q, r)\,[(l_1, \ldots, l_r)] ; l} \quad \text{(for-r)}$$

$P, P_1, P_2 \in \text{Prog}, \quad Q \in \text{Query}, \quad \sigma \in \textit{Context}, \quad y \in \textit{Variable},$
$r, i \in \mathbb{Z}_{\geq 0}, \quad x_1, \ldots, x_r \in \text{CRow}$

Fig. 11. Semantics for executing a program using a context to obtain a loop layout tree

(Figure 6b). 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 $\boxed{P}$ and configures an initial context σ where all database tables are empty and the input parameters are distinct. It invokes GetTrace on $\boxed{P}$ 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

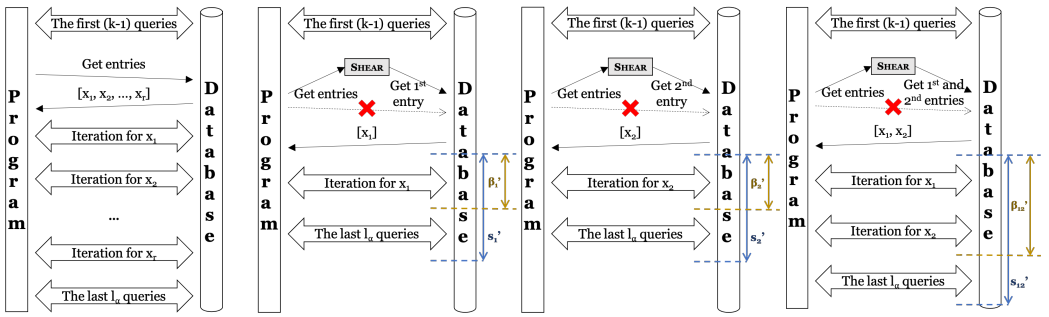
Input: $\boxed{P}$ is the executable of $P \in \mathcal{S}$.
Output: Program equivalent to P.

1: **procedure** INFER($\boxed{P}$)
2: $\sigma \leftarrow$ Database empty, input parameters distinct ; $t \leftarrow$ GETTRACE($\boxed{P}$, Nil, σ) ; **return** INFERPROG($\boxed{P}$, Nil, t)
3: **end procedure**

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

Input: $\boxed{P}$ is the executable of $P \in \mathcal{S}$. W is a path constraint. σ is a context that satisfies W.
Output: Annotated trace t, $t \sim W$, from executing $\boxed{P}$.

1: **procedure** GETTRACE($\boxed{P}$, W, σ)
2: $e \leftarrow$ EXECUTE($\boxed{P}$, σ) ; $l \leftarrow$ DETECTLOOPS($\boxed{P}$, σ, e) ; **return** MATCHPATH(l, W)
3: **end procedure**



(a) Intact trace (b) First altered trace (c) Second altered trace (d) Third altered trace

Fig. 12. Illustration of Algorithm 4 when applied to a true loop that iterates over the k-th query

(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 $\boxed{P}$ and a path constraint W. The procedure solves for a context σ that enables $\boxed{P}$ to produce a trace t that satisfies W. If satisfiable, the procedure invokes 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: $\boxed{P}$ is the executable of a program $P \in \mathcal{S}$.
Input: σ is a context.
Input: k is an integer.
Input: ρ is a list of distinct integers.
Output: List of SQL queries obtained from executing $\boxed{P}$ with σ, altering the k-th query to retrieve only the rows specified in ρ.

```
1:  procedure ExecAndPick(P, σ, k, ρ)
2:      s ← Empty list
3:      ⟨σ_I, σ_D, σ_R⟩ ← σ
4:      Populate the database with contents σ_D
5:      Execute P with input parameters σ_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 ← SQL query retrieving only rows ρ in q
10:         end if
11:         Send q to database
12:         d ← Data retrieved from the database
13:         Send d to P
14:         Append q to s
15:     end for
16:     return s
17: 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 ρ. 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: $\boxed{P}$ is the executable of $P \in \mathbb{S}$. σ 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 $\boxed{P}$ with σ.
Output: Integer l_α represents the number of queries in the trace produced by the subprogram that follows the detected loop.

1: **procedure** DetectHypo($\boxed{P}, \sigma, k$)
2: $s_1 \leftarrow$ ExecAndPick($\boxed{P}, \sigma, k, [1]$) ; $s_2 \leftarrow$ ExecAndPick($\boxed{P}, \sigma, k, [2]$) ; $s_{12} \leftarrow$ ExecAndPick($\boxed{P}, \sigma, k, [1, 2]$)
3: $s'_1 \leftarrow s_1[k+1, \ldots]$; $s'_2 \leftarrow s_2[k+1, \ldots]$; $s'_{12} \leftarrow s_{12}[k+1, \ldots]$
4: $l_\alpha \leftarrow \text{Len}(s'_1) + \text{Len}(s'_2) - \text{Len}(s'_{12})$ ▷ Len. after hypo. loop
5: $l_1 \leftarrow \text{Len}(s'_1) - l_\alpha$ ▷ Len. of the first hypo. iteration
6: $l_2 \leftarrow \text{Len}(s'_2) - l_\alpha$ ▷ Len. of the second hypo. iteration
7: $l_{12} \leftarrow \text{Len}(s'_{12}) - l_\alpha$ ▷ Len. of both hypo. iterations
8: $\beta_1 \leftarrow s'_1[1, \ldots, l_1]$ ▷ Queries in the first hypo. iteration
9: $\beta_{12} \leftarrow s'_{12}[1, \ldots, l_{12}]$ ▷ Queries in both hypo. iterations
10: $\alpha_1 \leftarrow s'_1[l_1 + 1, \ldots]$ ▷ Queries after the hypo. loop
11: $\alpha_2 \leftarrow s'_2[l_2 + 1, \ldots]$ ▷ Queries after the hypo. loop
12: $\alpha_{12} \leftarrow s'_{12}[l_{12} + 1, \ldots]$ ▷ Queries after the hypo. loop
13: **if** $\alpha_1 = \alpha_2 = \alpha_{12}$ **and** PerfectPrefix(β_1, β_{12}) **then**
14: **return** $\langle$`true`, $l_\alpha\rangle$ ▷ Traces consistent with hypo. loop
15: **end if**
16: **return** $\langle$`false`, Nil$\rangle$ ▷ Traces inconsistent with hypo. loop
17: **end procedure**

specified by the row indices in ρ. 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 σ. Executing the program with the context σ 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 σ 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 σ 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 β_1 and β_{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_α 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).

Algorithm 5 Identify iteration lengths

Input: $\boxed{P}$ is the executable of $P \in \mathcal{S}$. σ is a context. k, r, l_α are integers denoting a loop at k-th query, the number of iterations, and the number of queries after loop, respectively.
Output: List l_β of r integers, where the i-th ($i = 1, \ldots, r$) integer represents the number of queries produced by the i-th loop iteration.

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

Algorithm 6 Detect all loops

Input: $\boxed{P}$ is the executable of $P \in \mathcal{S}$. σ is a context. e is the list of query-result pairs obtained from executing P with σ.
Output: Loop layout tree constructed from e.

1: **procedure** DetectLoops($\boxed{P}, \sigma, e$)
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: $\langle f, l_\alpha \rangle \leftarrow$ DetectHypo($\boxed{P}, \sigma, k$)
7: **if not** f **then continue end if**
8: $l_\beta \leftarrow$ DetectIters($\boxed{P}, \sigma, k, r_k, l_\alpha$)
9: Append $\langle k, l_\beta \rangle$ to L
10: **end for**
11: **return** BuildLLTree(e, L)
12: **end procedure**

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 σ, and three integers k, r, l_α. Shear invokes this procedure only when it

detects a loop that iterates over the r rows retrieved by the k-th query in the trace from executing the program with σ. 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_α 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. [1] Each invocation executes the program with context σ, 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_α queries that are produced by the subprogram in P after the loop. Figure 13 illustrates these calculations.

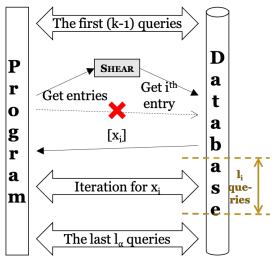


Fig. 13. Illustration of Alg. 5

The procedure calculates the lengths of each loop iteration, then returns all of these lengths as a list l_β. 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 $\sigma \in \mathit{Context}$, $\sigma[\mapsto_{:k} P]$ denotes the updated context after evaluating P in σ 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.

$$\frac{}{\sigma[\mapsto_{:0} P] = \sigma} \quad \text{(zero)}$$

$$\frac{\sigma[Q.y \mapsto \sigma(Q)][\mapsto_{:k} P] = \sigma'}{\sigma[\mapsto_{:k+1} Q\ P] = \sigma'} \quad \text{(seq)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma[\mapsto_{:k} P_2] = \sigma'}{\sigma[\mapsto_{:k+1} \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2] = \sigma'} \quad \text{(if-0)}$$

$$\frac{|\sigma(Q)| > 0 \qquad \sigma[Q.y \mapsto \sigma(Q)][\mapsto_{:k} P_1] = \sigma'}{\sigma[\mapsto_{:k+1} \texttt{if}\ Q\ \texttt{then}\ P_1\ \texttt{else}\ P_2] = \sigma'} \quad \text{(if-1)}$$

$$\frac{|\sigma(Q)| = 0 \qquad \sigma[\mapsto_{:k} P_2] = \sigma'}{\sigma[\mapsto_{:k+1} \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2] = \sigma'} \quad \text{(for-0)}$$

$$\frac{\begin{array}{c}\sigma(Q) = (x_1, \ldots, x_r) \qquad r > 0 \qquad \sigma[\mapsto_{:k} P_2] = \sigma' \\ \textsc{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_i]) = k_i \quad \text{for all } i = 1, \ldots, r\end{array}}{\sigma[\mapsto_{:k_1+\ldots+k_r+k+1} \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2] = \sigma'} \quad \text{(for-r)}$$

$$\frac{\begin{array}{c}\sigma(Q) = (x_1, \ldots, x_r) \qquad 0 < j \le r \\ \textsc{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_i]) = k_i \quad \text{for all } i = 1, \ldots, j-1 \\ \textsc{Len}_{\text{exec}}(P_1, \sigma[Q.y \mapsto x_j]) > k_j \qquad \sigma[Q.y \mapsto x_j][\mapsto_{:k_j} P_1] = \sigma'\end{array}}{\sigma[\mapsto_{:k_1+\ldots+k_j+1} \texttt{for}\ Q\ \texttt{do}\ P_1;\ P_2] = \sigma'} \quad \text{(for-j)}$$

$P, P_1, P_2 \in \text{Prog}, \quad Q \in \text{Query}, \quad \sigma, \sigma' \in \textit{Context}, \quad y \in \textit{Variable},$
$r, i, j, k, k_1, \ldots, k_r \in \mathbb{Z}_{\ge 0}, \quad x_1, \ldots, x_r \in \text{CRow}$

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

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

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 $\sigma \in \textit{Context}$, a loop iterates over the k-th query for r_k times if the following hold for some $P_1, P_2 \in \text{Prog}$: $\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n)$, $1 \le k \le n$, and $P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for}\ Q_k\ \texttt{do}\ P_1;\ P_2$. In this case, $\sigma \vdash_{k:} P \Downarrow_{\text{before}} e$ denotes the list of query-result pairs e produced by the subprogram before the loop. $\sigma \vdash_{k:} P \Downarrow^{i}_{\text{iter}} e_i$ denotes the list of query-result pairs e_i obtained from evaluating the i-th iteration of the loop ($i = 1, \ldots, r_k$). $\sigma \vdash_{k:} P \Downarrow_{\text{after}} e$ 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. $\textsc{Len}_{\text{iter}}(P, \sigma, k, i)$ denotes the number of queries produced by the i-th iteration of the loop, that is, $\textsc{Len}_{\text{iter}}(P, \sigma, k, i) = \textsc{Len}(e_i)$ where $\sigma \vdash_{k:} P \Downarrow^{i}_{\text{iter}} e_i$ ($i = 1, \ldots, r_k$).

Definition 14. For program $P \in \text{Prog}$, context $\sigma \in \textit{Context}$, and integer k, $P[k]_\sigma$ denotes the k-th query evaluated when executing P in σ. Figure 18 defines this concept.

$$\frac{}{P \;-\; \text{Nil} \;=\; P} \quad \text{(nil)}$$

$$\frac{P \;-\; e \;=\; P'}{Q\; P \;-\; (Q, r)\; @\, e \;=\; P'} \quad \text{(seq)}$$

$$\frac{P_2 \;-\; e \;=\; P_2'}{\texttt{if}\; Q\; \texttt{then}\; P_1\; \texttt{else}\; P_2 \;-\; (Q, 0)\; @\, e \;=\; P_2'} \quad \text{(if-0)}$$

$$\frac{r > 0 \qquad P_1 \;-\; e \;=\; P_1'}{\texttt{if}\; Q\; \texttt{then}\; P_1\; \texttt{else}\; P_2 \;-\; (Q, r)\; @\, e \;=\; P_1'} \quad \text{(if-1)}$$

$$\frac{P_2 \;-\; e \;=\; P_2'}{\texttt{for}\; Q\; \texttt{do}\; P_1;\; P_2 \;-\; (Q, 0)\; @\, e \;=\; P_2'} \quad \text{(for-0)}$$

$$\frac{r > 0 \qquad P_1 \;-\; e_i \;=\; \epsilon \quad \text{for all } i = 1, \ldots, r \qquad P_2 \;-\; e \;=\; P_2'}{\texttt{for}\; Q\; \texttt{do}\; P_1;\; P_2 \;-\; (Q, r)\; @\, e_1\; @\; \ldots\; @\, e_r\; @\, e \;=\; P_2'} \quad \text{(for-r)}$$

$$\frac{\begin{array}{c} 0 < j \le r \qquad e = (Q, r)\; @\, e_1\; @\; \ldots\; @\, e_j \\ P_1 \;-\; e_i \;=\; \epsilon \quad \text{for all } i = 1, \ldots, j-1 \qquad P_1 \;-\; e_j \;=\; P_1' \neq \epsilon \end{array}}{\texttt{for}\; Q\; \texttt{do}\; P_1;\; P_2 \;-\; e \;=\; \textsc{ConnectProgs}(P_1', \underbrace{P_1, \ldots, P_1}_{(r-j)\text{ times}}, P_2)} \quad \text{(for-j)}$$

$$P, P_1, P_2, P', P_1', P_2' \in \text{Prog}, \quad Q \in \text{Query}, \quad r, i, j \in \mathbb{Z}_{\ge 0}$$

Fig. 15. Use a list of query-result pairs to consume a program prefix and obtain the remaining subprogram

Definition 15. For program $P \in \text{Prog}$, context $\sigma \in \mathit{Context}$, integer k where a loop iterates over the k-th query, and list of integers $\rho_1, \ldots, \rho_m$, $\sigma \vdash_{k:} P \Downarrow_{\text{alter}}^{\rho_1, \ldots, \rho_m} e$ denotes the list of query-result pairs e produced by an altered evaluation of P in σ 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.

Proposition 1. For program $P \in \mathcal{S}$, context $\sigma \in \mathit{Context}$, and integer k, if $\Lambda(P, \sigma, k) = [e_1, \ldots, e_r]$, $\sigma \vdash_{k:} P \Downarrow_{\text{alter}}^{\text{Nil}} (Q_1, r_1), \ldots, (Q_n, r_n)$, and $\textsc{ExecAndPick}(\boxed{P}, \sigma, k, []) = [q_1, \ldots, q_{n'}]$, 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 $\sigma \in \mathit{Context}$, integer k, and list of distinct integers $\rho_1, \ldots, \rho_m$ $(m > 0)$, if $\Lambda(P, \sigma, k) = [e_1, \ldots, e_r]$, $1 \le \rho_1 < \ldots < \rho_m \le r$, $\sigma \vdash_{k:} P \Downarrow_{\text{alter}}^{\rho_1, \ldots, \rho_m} (Q_1, r_1), \ldots, (Q_n, r_n)$, and $\textsc{ExecAndPick}(\boxed{P}, \sigma, k, [\rho_1, \ldots, \rho_m]) = [q_1, \ldots, q_{n'}]$, 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 $\sigma \in \mathit{Context}$, and integer k, if we have $\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n)$, $1 \le k \le n$, $r_k \ge 2$, and $\Lambda(P, \sigma, k) = [e_1, \ldots, e_{r_k}]$, then we have $\textsc{DetectHypo}(\boxed{P}, \sigma, k) = \langle \texttt{true}, \textsc{Len}(e') \rangle$ where $\sigma \vdash_{k:} P \Downarrow_{\text{after}} e'$.

Proof Sketch. By induction on k and the derivation of loop body. □

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for } Q_k \texttt{ do } P_1; P_2}{\sigma \vdash_{k:} P \Downarrow_{\text{before}} (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1})} \quad \text{(before)}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for } Q_k \texttt{ do } P_1; P_2 \\ \sigma[\mapsto_{:k} P] \vdash P_1 \Downarrow_{\text{exec}} e}{\sigma \vdash_{k:} P \Downarrow^1_{\text{iter}} e} \quad \text{(iter-1)}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for } Q_k \texttt{ do } P_1; P_2 \\ l_i = \text{LEN}_{\text{iter}}(P, \sigma, k, i) \quad \text{for all } i = 1, \ldots, j-1 \\ 2 \le j \le r_k \qquad \sigma[\mapsto_{:k+l_1+\ldots+l_{j-1}} P] \vdash P_1 \Downarrow_{\text{exec}} e}{\sigma \vdash_{k:} P \Downarrow^j_{\text{iter}} e} \quad \text{(iter-j)}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for } Q_k \texttt{ do } P_1; P_2 \\ \sigma[\mapsto_{:k-1} P] \vdash P_2 \Downarrow_{\text{exec}} e}{\sigma \vdash_{k:} P \Downarrow_{\text{after}} e} \quad \text{(after)}$$

$P, P_1, P_2, \in \text{Prog}, \quad Q, Q_1, \ldots, Q_n \in \text{Query}, \quad \sigma \in \textit{Context}, \\ n, i, j, k, r_1, \ldots, r_n, l_1, \ldots, l_{j-1} \in \mathbb{Z}_{\ge 0}$

Fig. 16. Calculate query-result pairs that correspond to evaluating the subprograms before a loop, in each loop iteration, and after the loop

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = Q_k\ P_1}{\Lambda(P, \sigma, k) = \text{NotLoop}} \quad \text{(seq)}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{if } Q_k \texttt{ then } P_1 \texttt{ else } P_2}{\Lambda(P, \sigma, k) = \text{NotLoop}} \quad \text{(if)}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \\ P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for } Q_k \texttt{ do } P_1; P_2 \\ \sigma \vdash_{k:} P \Downarrow^j_{\text{iter}} e_j \quad \text{for all } j = 1, \ldots, r_k}{\Lambda(P, \sigma, k) = [e_1, \ldots, e_{r_k}]} \quad \text{(for)}$$

$P, P_1, P_2, \in \text{Prog}, \quad Q, Q_1, \ldots, Q_n \in \text{Query}, \quad \sigma \in \textit{Context}, \\ n, j, k, r_1, \ldots, r_n \in \mathbb{Z}_{\ge 0}$

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

Theorem 2. For any program $P \in \mathcal{S}$, context $\sigma \in \textit{Context}$, and integer k, if $\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n)$, $1 \le k \le n$, $r_k \ge 2$, and $\Lambda(P, \sigma, k) = \text{NotLoop}$, then $\text{DETECTHYPO}(\boxed{P}, \sigma, k) = \langle \texttt{false}, \text{Nil} \rangle$.

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \qquad P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = Q_k\ P_1}{P[k]_\sigma = Q_k} \tag{seq}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \qquad P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{if}\ Q_k\ \texttt{then}\ P_1\ \texttt{else}\ P_2}{P[k]_\sigma = Q_k} \tag{if}$$

$$\frac{\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n) \qquad 1 \le k \le n \qquad P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = \texttt{for}\ Q_k\ \texttt{do}\ P_1;\ P_2}{P[k]_\sigma = Q_k} \tag{for}$$

$P, P_1, P_2, \in \text{Prog}, \quad Q, Q_1, \ldots, Q_n \in \text{Query}, \quad \sigma \in \mathit{Context}, \quad n, k, r_1, \ldots, r_n \in \mathbb{Z}_{\ge 0}$

Fig. 18. The k-th query evaluated when executing a program

$$\frac{\Lambda(P, \sigma, k) = [e_1, \ldots, e_r] \qquad P[k]_\sigma = Q \qquad \sigma \vdash_{k:} P \Downarrow_{\text{before}} e' \qquad \sigma \vdash_{k:} P \Downarrow_{\text{after}} e''}{\sigma \vdash_{k:} P \Downarrow^{\text{Nil}}_{\text{alter}} e'\ @\ (Q, 0)\ @\ e''} \tag{nil}$$

$$\frac{\Lambda(P, \sigma, k) = [e_1, \ldots, e_r] \qquad P[k]_\sigma = Q \qquad 1 \le \rho_i \le r \text{ and } \sigma \vdash_{k:} P \Downarrow^{\rho_i}_{\text{iter}} e_i \quad \text{for all } i = 1, \ldots, m \qquad \sigma \vdash_{k:} P \Downarrow_{\text{before}} e' \qquad \sigma \vdash_{k:} P \Downarrow_{\text{after}} e''}{\sigma \vdash_{k:} P \Downarrow^{\rho_1, \ldots, \rho_m}_{\text{alter}} e'\ @\ (Q, m)\ @\ e_1\ @\ \ldots\ @\ e_m\ @\ e''} \tag{rows}$$

$P \in \text{Prog}, \quad \sigma \in \mathit{Context}, \quad r, k, i, m, \rho_1, \ldots, \rho_m \in \mathbb{Z}_{\ge 0}$

Fig. 19. Semantics for executing a program while altering a query to retrieve only the specified rows

Proof Sketch. The proof performs a case analysis of whether a subprogram P' (see below) references Q_k and, if so, where is the first reference. Here P' satisfies $P - (Q_1, r_1), \ldots, (Q_{k-1}, r_{k-1}) = P'$. □

Theorem 3. For any program $P \in \mathcal{S}$, context $\sigma \in \mathit{Context}$, and integer k, if $\Lambda(P, \sigma, k) = [e_1, \ldots, e_r]$ and $\sigma \vdash_{k:} P \Downarrow_{\text{after}} e'$, then DetectLoopIters($\boxed{P}$, σ, k, r, Len(e')) = [Len(e_1), ..., Len(e_r)].

Theorem 4. For any program $P \in \mathcal{S}$ and context $\sigma \in \mathit{Context}$, if $\sigma \vdash P \Downarrow_{\text{exec}} e$ and $\sigma \vdash P \Downarrow_{\text{loops}} l$, then DetectLoops($\boxed{P}$, σ, 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 INFER 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 $\boxed{P}$ 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

Algorithm 7 Recursively infer a subprogram

Input: $\boxed{P}$ is the executable of a program $P \in \mathcal{S}$. s_1, s_2 are a prefix and a suffix of an annotated trace.
Output: Subprogram equivalent to P's subprogram after trace s_1.

procedure InferProg($\boxed{P}$, s_1, s_2)
  if s_2 = Nil **then return** ϵ **end if** ▷ Prog := ϵ
  $k \leftarrow$ The length of s_1 ; $Q \leftarrow$ The first query in s_2 ; $d, a \leftarrow$ GetKnownLoops(s_1)
  for $i = 0, 1, 2$ **do**
    $W_i \leftarrow$ MakePathConstraint($s_1, Q, i, (d$ @ **false**$), (a$ @ **false**$)$)
    $(f_i, t_i) \leftarrow$ SolveAndGetTrace($\boxed{P}$, W_i)
    if f_i **then** $t_{i,1} \leftarrow t_i[1, \ldots, (k+1)]$; $t_{i,2} \leftarrow t_i[(k+2), \ldots]$ **end if** ▷ Satisfiable
  end for
  if f_2 **and** found loop on the last query in $t_{2,1}$ **then**
    $W_2' \leftarrow$ MakePathConstraint($s_1, Q, 2, (d$ @ **true**$), (a$ @ **false**$)$) ▷ Loop body
    $W_0' \leftarrow$ MakePathConstraint($s_1, Q, Nil, (d$ @ **true**$), (a$ @ **true**$)$) ▷ After-loop subprogram
    $(f_2', t_2') \leftarrow$ SolveAndGetTrace($\boxed{P}$, W_2') ; $(f_0', t_0') \leftarrow$ SolveAndGetTrace($\boxed{P}$, W_0')
    $b_{\text{iter}} \leftarrow$ InferProg($\boxed{P}$, $t_2'[1, \ldots, (k+1)], t_2'[(k+2), \ldots]$)
    $b_{\text{after}} \leftarrow$ InferProg($\boxed{P}$, $t_0'[1, \ldots, (k+1)], t_0'[(k+2), \ldots]$)
    return "`for` Q `do` b_{iter} ; b_{after}" ▷ Prog := For
  else if f_0 **and** f_1 **and** (($t_{0,2} \neq$ Nil **and** $t_{1,2} \neq$ Nil) **or** ($t_{0,2} \neq$ Nil **and** $t_{1,2} =$ Nil) **or** the first queries in $t_{0,2}$ and $t_{1,2}$ have different skeletons) **then**
    $b_t \leftarrow$ InferProg($\boxed{P}$, $t_{1,1}, t_{1,2}$) ; $b_f \leftarrow$ InferProg($\boxed{P}$, $t_{0,1}, t_{0,2}$)
    return "`if` Q `then` b_t `else` b_f" ▷ Prog := If
  else
    if f_0 **then** $b \leftarrow$ InferProg($\boxed{P}$, $t_{0,1}, t_{0,2}$) **else** $b \leftarrow$ InferProg($\boxed{P}$, $t_{1,1}, t_{1,2}$) **end if**
    return "Q b" ▷ Prog := Seq
  end if
end procedure

statement, but not both branches. 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.

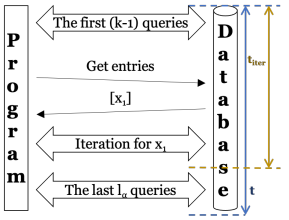
To achieve this goal, Shear repurposes its loop detection algorithm to identify the boundary of the (only one) loop iteration in the trace. Specifically, Shear first matches a trace t against the known Prog nonterminals that have already been inferred. If a query Q is known to be generated by a For statement 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 ρ. 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_{after} contains only the after-loop queries, without any loop-body queries. Next, SHEAR uses t_{after} to locate the boundary of the loop body in t and discards all of the subsequent queries. The resulting trace t_{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, SHEAR is able to discard t and replace it with two new traces t_{iter} and t_{after} that correspond precisely to the loop body and the after-loop subprogram, respectively.

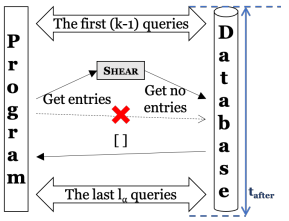
5.3 Soundness Proof Outline

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 $\mathcal{S}$ 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 INFERPROG procedure.

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



(a) Intact trace



(b) Altered trace

Fig. 20. Detecting boundaries of the only iteration

5.4 Discussion

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 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, SHEAR supports a wide range of programs in the SHEAR DSL.

A fundamental enhancement of 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 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 SHEAR eliminates all of these restrictions, enabling SHEAR to correctly infer a larger range of programs (Section 6). In other words, despite supporting a more expressive DSL, 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.

```
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

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

(b) with q non-loop repetitive queries

```
def rec(n_layer):
  if n_layer <= 0:
    do_sql("SELECT * FROM t0")
    return
  s = do_sql("SELECT * FROM t1")
  for row in s:
    rec(n_layer - 1)

rec(n)
```

(c) with n layers of nested loops

Fig. 21. Programs for testing scalability

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.

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), 1–17 queries (Figure 22c), 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; Biermann 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

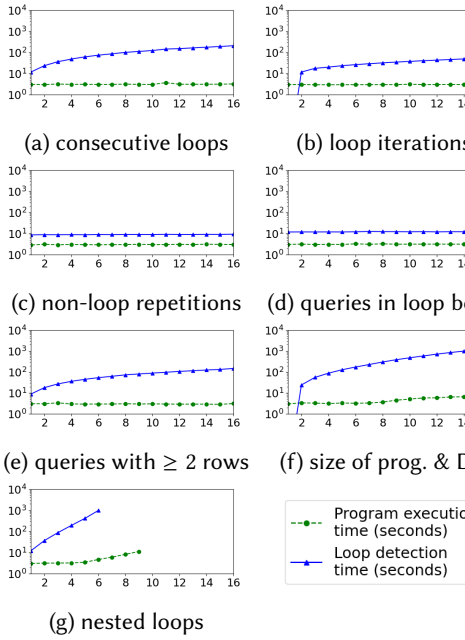


Fig. 22. Time to detect loops in programs with various measures of complexity

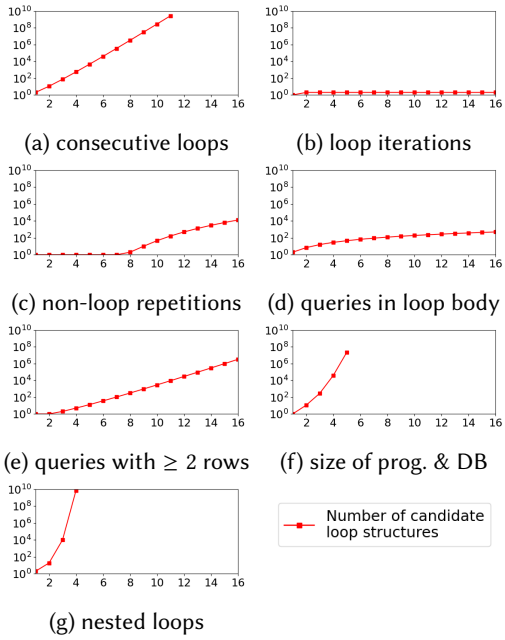


Fig. 23. Number of candidate loop structures in programs with various measures of complexity

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 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: $N_{k+1} = N_k + (N_k^2 - 1) + (N_k - 1)N_k + 1$, 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 ϵ, 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.

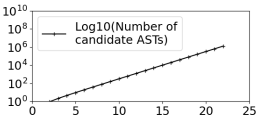


Fig. 24. Log base 10 of the number of candidate AST structures in programs with various maximum depths

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 180 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 800 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.

Table 1. Comparison of Shear and other loop detection techniques

Command	App	Time	Q	For	Struct	S	Kn	D	Kb
get_projects_id_messages	RailsCollab	7203.3s	34	3	R, C, A_2	✓	×	×	×
get_projects_id_messages_id	RailsCollab	4326.4s	21	0	r	✓	×	×	×
get_projects_id_messages_display_list	RailsCollab	7461.4s	35	3	R, C, A_2	✓	×	×	×
get_projects_id_times (fixed 500 error)	RailsCollab	8739.7s	38	1	R, C, A	✓	×	×	×
get_projects_id_times_id	RailsCollab	3846.7s	55	0	R	✓	×	✓	×
get_projects_id_milestones_id	RailsCollab	5544.6s	25	2	R, r, A	✓	×	×	×
get_users_id (fixed 500 error)	RailsCollab	960.3s	12	1		✓	✓	✓	✓
get_api_lists	Kanban	1557.2s	9	3	C, N_3	✓	×	×	×
get_api_lists_id	Kanban	765.5s	9	2	C, N_2	✓	×	×	×
get_api_cards	Kanban	780.3s	8	2	C, N_2	✓	×	×	×
get_api_cards_id	Kanban	398.4s	8	1		✓	✓	✓	✓
get_api_boards_id	Kanban	1329.2s	10	3	C, N_3, A	✓	×	×	×
get_home	Todo	529.3s	5	1	C	✓	✓	×	×
get_lists_id_tasks	Todo	565.9s	6	1	C, A	✓	×	×	×
get_lists_id_tasks (fixed 404 error)	Todo	742.2s	6	1	C, A	✓	×	×	×
get_channels	Kandan	3762.2s	16	2	C	✓	✓	×	×
get_home	Enki	757.2s	9	1	A	✓	×	✓	✓
get_archives	Enki	659.4s	6	1	C, A	✓	×	×	×
get_admin_posts	Enki	397.5s	3	1		✓	✓	✓	✓
get_admin (trimmed)	Enki	773.9s	7	2	A_2	✓	×	✓	✓
liststudentcourses	Student	171.1s	5	1		✓	✓	✓	✓
repeat_2	Synthetic	82.0s	3	0	r	✓	×	×	×
repeat_3	Synthetic	102.7s	4	0	R, r	✓	×	×	×
repeat_4	Synthetic	122.2s	5	0	R, r	✓	×	×	×
repeat_5	Synthetic	143.0s	6	0	R, r	✓	×	×	×
nest	Synthetic	393.8s	3	2	C, N_2	✓	×	×	×
after_2	Synthetic	220.8s	4	2	R, A_2	✓	×	✓	×
after_3	Synthetic	328.7s	6	3	R, A_3	✓	×	✓	×
after_4	Synthetic	441.1s	8	4	R, A_4	✓	×	✓	×
after_5	Synthetic	549.4s	10	5	R, A_5	✓	×	✓	×
example (Section 2)	Synthetic	293.3s	6	1	R, A	✓	×	✓	×

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

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 [3] 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

[3] For 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; Rodríguez 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-Broştean 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

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

Table 3. Summary of the programs regenerated by SHEAR

Command	App	In	Q	If	For	Out	LoC	Comp
get_projects_id_messages	RailsCollab	2	34	9	3	39	114	75
get_projects_id_messages_id	RailsCollab	3	21	5	0	30	69	39
get_projects_id_messages_display_list	RailsCollab	2	35	9	3	40	115	75
get_projects_id_times (fixed 500 error)	RailsCollab	2	38	11	1	40	114	74
get_projects_id_times_id	RailsCollab	3	55	15	0	50	147	97
get_projects_id_milestones_id	RailsCollab	3	25	6	2	41	93	52
get_projects	RailsCollab	1	7	2	0	9	26	17
get_companies_id	RailsCollab	2	6	2	0	12	28	16
get_users_id (fixed 500 error)	RailsCollab	2	12	5	1	18	53	35
get_api_lists	Kanban	1	9	1	3	27	55	28
get_api_lists_id	Kanban	2	9	2	2	27	54	27
get_api_cards	Kanban	1	8	1	2	23	46	23
get_api_cards_id	Kanban	2	8	2	1	23	45	22
get_api_boards_id	Kanban	2	10	2	3	33	64	31
get_api_users_current	Kanban	1	1	0	0	4	9	5
get_api_users_id	Kanban	2	1	0	0	4	9	5
get_home	Todo	0	5	1	1	5	20	15
get_lists_id_tasks	Todo	1	6	1	1	2	17	15
get_lists_id_tasks (fixed 404 error)	Todo	1	6	1	1	5	20	15
get_home	Fulcrum	1	5	1	0	9	21	12
get_projects	Fulcrum	1	5	1	0	9	21	12
get_projects_id	Fulcrum	2	8	2	0	8	25	17
get_projects_id_stories	Fulcrum	2	8	3	0	11	31	20
get_projects_id_stories_id	Fulcrum	3	9	3	0	11	31	20
get_projects_id_stories_id_notes	Fulcrum	3	9	3	0	4	24	20
get_projects_id_stories_id_notes_id	Fulcrum	4	10	4	0	4	28	24
get_projects_id_users	Fulcrum	2	8	2	0	8	25	17
get_channels	Kandan	1	16	4	2	15	53	38
get_channels_id_activities	Kandan	2	16	6	0	13	49	36
get_channels_id_activities_id	Kandan	3	11	3	0	3	25	22
get_me	Kandan	1	8	3	0	25	44	19
get_users	Kandan	1	11	3	0	45	67	22
get_users_id	Kandan	2	8	3	0	25	44	19
get_home	Enki	0	9	1	1	8	26	18
get_archives	Enki	0	6	1	1	5	21	16
get_admin_comments_id	Enki	1	1	0	0	5	10	5
get_admin_pages	Enki	0	2	1	0	4	13	9
get_admin_pages_id	Enki	1	1	0	0	4	9	5
get_admin_posts	Enki	0	3	1	1	3	17	14
get_admin (trimmed)	Enki	0	7	0	2	3	21	18
get_article_id	Blog	1	2	1	0	6	15	9
get_articles	Blog	0	1	0	0	3	8	5
liststudentcourses	Student	2	5	2	1	3	22	19
repeat_2	Synthetic	0	3	0	0	0	7	7
repeat_3	Synthetic	0	4	0	0	0	8	8
repeat_4	Synthetic	0	5	0	0	0	9	9
repeat_5	Synthetic	0	6	0	0	0	10	10
nest	Synthetic	0	3	0	2	0	15	15
after_2	Synthetic	0	4	0	2	0	15	15
after_3	Synthetic	0	6	0	3	0	20	20
after_4	Synthetic	0	8	0	4	0	25	25
after_5	Synthetic	0	10	0	5	0	30	30
example (Section 2)	Synthetic	1	6	1	1	6	22	16

REFERENCES

2018. Enki: A Ruby on Rails blogging app for the fashionable developer. https://github.com/xaviershay/enki.

2018. Fulcrum: An agile project planning tool. https://github.com/fulcrum-agile/fulcrum.

2018. Getting Started with Rails. http://guides.rubyonrails.org/getting_started.html.

2018. Kandan – Modern Open Source Chat. https://github.com/kandanapp/kandan.

2019. Proxy - JavaScript | MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy.

2020. Software Assurance Reference Dataset. https://samate.nist.gov/SARD/testsuite.php.

2021. Kanban: a Trello clone in Rails and Backbone.js. https://github.com/somlor/kanban.

2021. RailsCollab: A Project Management and Collaboration tool inspired by Basecamp. https://github.com/jamesu/railscollab/.

2021. Todo: Basic Rails GTD app. https://github.com/engineyard/todo.

F. Aarts, J. De Ruiter, and E. Poll. 2013. Formal Models of Bank Cards for Free. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. 461–468. https://doi.org/10.1109/ICSTW.2013.60

Fides Aarts and Frits Vaandrager. 2010. *Learning I/O Automata*. Springer Berlin Heidelberg, Berlin, Heidelberg, 71–85. https://doi.org/10.1007/978-3-642-15375-4_6

Maaz Bin Safeer Ahmad and Alvin Cheung. 2018. Automatically Leveraging MapReduce Frameworks for Data-Intensive Applications. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1205–1220. https://doi.org/10.1145/3183713.3196891

Aws Albarghouthi, Sumit Gulwani, and Zachary Kincaid. 2013. Recursive Program Synthesis. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 934–950.

Saba Alimadadi, Sheldon Sequeira, Ali Mesbah, and Karthik Pattabiraman. 2014. Understanding JavaScript Event-Based Interactions. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 367–377. https://doi.org/10.1145/2568225.2568268

Kijin An and Eli Tilevich. 2020. Client Insourcing: Bringing Ops In-House for Seamless Re-Engineering of Full-Stack JavaScript Applications. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 179–189. https://doi.org/10.1145/3366423.3380105

Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (Nov. 1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

Osbert Bastani, Rahul Sharma, Alex Aiken, and Percy Liang. 2017. Synthesizing program input grammars. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*. 95–110.

Osbert Bastani, Rahul Sharma, Alex Aiken, and Percy Liang. 2018. Active Learning of Points-to Specifications. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. ACM, New York, NY, USA, 678–692. https://doi.org/10.1145/3192366.3192383

A. W. Biermann, R. I. Baum, and F. E. Petry. 1975. Speeding Up the Synthesis of Programs from Traces. *IEEE Trans. Comput.* 24, 2 (Feb. 1975), 122–136. https://doi.org/10.1109/T-C.1975.224180

James F. Bowring, James M. Rehg, and Mary Jean Harrold. 2004. Active Learning for Automatic Classification of Software Behavior. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis* (Boston, Massachusetts, USA) *(ISSTA '04)*. ACM, New York, NY, USA, 195–205. https://doi.org/10.1145/1007512.1007539

M. Burtscher, I. Ganusov, S. J. Jackson, J. Ke, P. Ratanaworabhan, and N. B. Sam. 2005. The VPC trace-compression algorithms. *IEEE Trans. Comput.* 54, 11 (Nov 2005), 1329–1344. https://doi.org/10.1109/TC.2005.186

Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. 2009. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-engineering. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '09)*. ACM, New York, NY, USA, 621–634. https://doi.org/10.1145/1653662.1653737

Juan Caballero and Dawn Song. 2013. Automatic protocol reverse-engineering: Message format extraction and field semantics inference. *Computer Networks* 57, 2 (2013), 451 – 474. https://doi.org/10.1016/j.comnet.2012.08.003 Botnet Activity: Analysis, Detection and Shutdown.

Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. 2007. Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (Alexandria, Virginia, USA) *(CCS '07)*. ACM, New York, NY, USA, 317–329. https://doi.org/10.1145/1315245.1315286

José P. Cambronero, Thurston H. Y. Dang, Nikos Vasilakis, Jiasi Shen, Jerry Wu, and Martin C. Rinard. 2019. Active Learning for Software Engineering. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Athens, Greece) *(Onward! 2019)*. Association for Computing Machinery, New York, NY, USA, 62–78. https://doi.org/10.1145/3359591.3359732

Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. 2011. Achievements and Challenges in Software Reverse Engineering. *Commun. ACM* 54, 4 (April 2011), 142–151. https://doi.org/10.1145/1924421.1924451

Michael Carbin, Sasa Misailovic, Michael Kling, and Martin C. Rinard. 2011. Detecting and escaping infinite loops with jolt. In *Proceedings of the 25th European conference on Object-oriented programming* (Lancaster, UK) *(ECOOP'11)*. Springer-Verlag, 609–633. http://dl.acm.org/citation.cfm?id=2032497.2032537

Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. 2016. Active learning for extended finite state machines. *Formal Aspects of Computing* 28, 2 (2016), 233–263. https://doi.org/10.1007/s00165-016-0355-5

Qiaochu Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. 2021. Web Question Answering with Neurosymbolic Program Synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 328–343. https://doi.org/10.1145/3453483.3454047

Qiaochu Chen, Xinyu Wang, Xi Ye, Greg Durrett, and Isil Dillig. 2020. Multi-Modal Synthesis of Regular Expressions. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 487–502. https://doi.org/10.1145/3385412.3385988

Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2014. Detecting Performance Anti-Patterns for Applications Developed Using Object-Relational Mapping. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 1001–1012. https://doi.org/10.1145/2568225.2568259

Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2016. Finding and Evaluating the Performance Impact of Redundant Data Access for Applications That Are Developed Using Object-Relational Mapping Frameworks. *IEEE Trans. Softw. Eng.* 42, 12 (Dec. 2016), 1148–1161. https://doi.org/10.1109/TSE.2016.2553039

Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. 2013. Optimizing Database-backed Applications with Query Synthesis. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. ACM, New York, NY, USA, 3–14. https://doi.org/10.1145/2491956.2462180

T. S. Chow. 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Softw. Eng.* 4, 3 (May 1978), 178–187. https://doi.org/10.1109/TSE.1978.231496

A. Cleve and J. Hainaut. 2008. Dynamic Analysis of SQL Statements for Data-Intensive Applications Reverse Engineering. In *2008 15th Working Conference on Reverse Engineering*. 192–196. https://doi.org/10.1109/WCRE.2008.38

Bas Cornelissen, Andy Zaidman, and Arie van Deursen. 2011. A Controlled Experiment for Program Comprehension through Trace Visualization. *IEEE Transactions on Software Engineering* 37, 3 (2011), 341–355. https://doi.org/10.1109/TSE.2010.47

Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, and Rainer Koschke. 2009. A Systematic Survey of Program Comprehension Through Dynamic Analysis. *IEEE Trans. Softw. Eng.* 35, 5 (Sept. 2009), 684–702. https://doi.org/10.1109/TSE.2009.28

Joeri De Ruiter and Erik Poll. 2015. Protocol State Fuzzing of TLS Implementations. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) *(SEC'15)*. USENIX Association, Berkeley, CA, USA, 193–206.

E. N. Elnozahy. 1999. Address Trace Compression Through Loop Detection and Reduction. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (Atlanta, Georgia, USA) *(SIGMETRICS '99)*. ACM, New York, NY, USA, 214–215. https://doi.org/10.1145/301453.301577

John K. Feser, Swarat Chaudhuri, and Isil Dillig. 2015. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*. 229–239.

Sheldon Finkelstein. 1982. Common Expression Analysis in Database Applications. In *Proceedings of the 1982 ACM SIGMOD International Conference on Management of Data* (Orlando, Florida) *(SIGMOD '82)*. Association for Computing Machinery, New York, NY, USA, 235–245. https://doi.org/10.1145/582353.582400

Paul Fiterău-Broştean, Ramon Janssen, and Frits Vaandrager. 2016. *Combining Model Learning and Model Checking to Analyze TCP Implementations*. Springer International Publishing, Cham, 454–471. https://doi.org/10.1007/978-3-319-41540-6_25

Timon Gehr, Dimitar Dimitrov, and Martin T. Vechev. 2015. Learning Commutativity Specifications. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. 307–323.

Georgios Giannikis, Gustavo Alonso, and Donald Kossmann. 2012. SharedDB: Killing One Thousand Queries with One Stone. *Proc. VLDB Endow.* 5, 6 (Feb. 2012), 526–537. https://doi.org/10.14778/2168651.2168654

Olga Grinchtein, Bengt Jonsson, and Martin Leucker. 2010. Learning of Event-recording Automata. *Theor. Comput. Sci.* 411, 47 (Oct. 2010), 4029–4054. https://doi.org/10.1016/j.tcs.2010.07.008

Sumit Gulwani. 2010. Dimensions in Program Synthesis. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (Hagenberg, Austria) *(PPDP '10)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/1836089.1836091

Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL '11)*. ACM, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

Stavros Harizopoulos, Vladislav Shkapenyuk, and Anastassia Ailamaki. 2005. QPipe: A Simultaneously Pipelined Relational Query Engine. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, Maryland) *(SIGMOD '05)*. Association for Computing Machinery, New York, NY, USA, 383–394. https://doi.org/10.1145/1066157.1066201

Hiroshige Hayashizaki, Peng Wu, Hiroshi Inoue, Mauricio J. Serrano, and Toshio Nakatani. 2011. Improving the Performance of Trace-based Systems by False Loop Filtering. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, California, USA) *(ASPLOS XVI)*. ACM, New York, NY, USA, 405–418. https://doi.org/10.1145/1950365.1950412

Stefan Heule, Manu Sridharan, and Satish Chandra. 2015. Mimic: computing models for opaque code. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. 710–720.

Malte Isberner, Falk Howar, and Bernhard Steffen. 2014. *The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning*. Springer International Publishing, Cham, 307–322. https://doi.org/10.1007/978-3-319-11164-3_26

Jinseong Jeon, Xiaokang Qiu, Jonathan Fetter-Degges, Jeffrey S. Foster, and Armando Solar-Lezama. 2016. Synthesizing framework models for symbolic execution. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*. 156–167.

Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided Component-based Program Synthesis. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. ACM, New York, NY, USA, 215–224. https://doi.org/10.1145/1806799.1806833

Ruyi Ji, Jingjing Liang, Yingfei Xiong, Lu Zhang, and Zhenjiang Hu. 2020a. Question Selection for Interactive Program Synthesis. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 1143–1158. https://doi.org/10.1145/3385412.3386025

Ruyi Ji, Yican Sun, Yingfei Xiong, and Zhenjiang Hu. 2020b. Guiding Dynamic Programing via Structural Probability for Accelerating Programming by Example. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 224 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428292

Shoaib Kamil, Alvin Cheung, Shachar Itzhaky, and Armando Solar-Lezama. 2016. Verified Lifting of Stencil Computations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. Association for Computing Machinery, New York, NY, USA, 711–726. https://doi.org/10.1145/2908080.2908117

Alain Ketterlin and Philippe Clauss. 2008. Prediction and Trace Compression of Data Access Addresses Through Nested Loop Recognition. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization* (Boston, MA, USA) *(CGO '08)*. ACM, New York, NY, USA, 94–103. https://doi.org/10.1145/1356058.1356071

Ravi Khadka, Belfrit V. Batlajery, Amir M. Saeidi, Slinger Jansen, and Jurriaan Hage. 2014. How Do Professionals Perceive Legacy Systems and Software Modernization?. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/2568225.2568318

Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. 2001. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP '01)*. Springer-Verlag, London, UK, UK, 327–353.

Michael Kling, Sasa Misailovic, Michael Carbin, and Martin Rinard. 2012. Bolt: on-demand infinite loop escape in unmodified binaries. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications* (Tucson, Arizona, USA) *(OOPSLA '12')*. ACM, 431–450. https://doi.org/10.1145/2384616.2384648

M. Kobayashi. 1984. Dynamic Characteristics of Loops. *IEEE Trans. Comput.* 33, 2 (Feb. 1984), 125–132. https://doi.org/10.1109/TC.1984.1676404

Paul Krogmeier, Umang Mathur, Adithya Murali, P. Madhusudan, and Mahesh Viswanathan. 2020. Decidable Synthesis of Programs with Uninterpreted Functions. In *Computer Aided Verification*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer International Publishing, Cham, 634–657.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. 2019. SPoC: Search-based Pseudocode to Code. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/7298332f04ac004a0ca44cc69ecf6f6b-Paper.pdf

Mina Lee, Sunbeom So, and Hakjoo Oh. 2016. Synthesizing Regular Expressions from Examples for Introductory Automata Assignments. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (Amsterdam, Netherlands) *(GPCE 2016)*. Association for Computing Machinery, New York, NY, USA, 70–80. https://doi.org/10.1145/2993236.2993244

Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. 2018. Accelerating Search-Based Program Synthesis Using Learned Probabilistic Models. *SIGPLAN Not.* 53, 4 (June 2018), 436–449. https://doi.org/10.1145/3296979.3192410

Josip Maras, Jan Carlson, and Ivica Crnkovi. 2012. Extracting Client-Side Web Application Code. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12)*. Association for Computing Machinery, New York, NY, USA, 819–828. https://doi.org/10.1145/2187836.2187947

Charith Mendis, Jeffrey Bosboom, Kevin Wu, Shoaib Kamil, Jonathan Ragan-Kelley, Sylvain Paris, Qin Zhao, and Saman Amarasinghe. 2015. Helium: Lifting High-performance Stencil Kernels from Stripped x86 Binaries to Halide DSL Code. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. ACM, New York, NY, USA, 391–402. https://doi.org/10.1145/2737924.2737974

Edward F Moore. 1956. Gedanken-experiments on sequential machines. *Automata studies* 34 (1956), 129–153.

Tipp Moseley, Daniel A. Connors, Dirk Grunwald, and Ramesh Peri. 2007. Identifying Potential Parallelism via Loop-centric Profiling. In *Proceedings of the 4th International Conference on Computing Frontiers* (Ischia, Italy) *(CF '07)*. ACM, New York, NY, USA, 143–152. https://doi.org/10.1145/1242531.1242554

Adrian Nistor, Linhai Song, Darko Marinov, and Shan Lu. 2013. Toddler: Detecting performance problems via similar memory-access patterns. In *2013 35th International Conference on Software Engineering (ICSE)*. 562–571. https://doi.org/10.1109/ICSE.2013.6606602

Nesrine Noughi, Marco Mori, Loup Meurice, and Anthony Cleve. 2014. Understanding the Database Manipulation Behavior of Programs. In *Proceedings of the 22Nd International Conference on Program Comprehension* (Hyderabad, India) *(ICPC 2014)*. ACM, New York, NY, USA, 64–67. https://doi.org/10.1145/2597008.2597790

Peter-Michael Osera and Steve Zdancewic. 2015. Type-and-Example-Directed Program Synthesis. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. Association for Computing Machinery, New York, NY, USA, 619–630. https://doi.org/10.1145/2737924.2738007

Shankara Pailoor, Yuepeng Wang, Xinyu Wang, and Isil Dillig. 2021. Synthesizing Data Structure Refinements from Integrity Constraints. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 574–587. https://doi.org/10.1145/3453483.3454063

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. 2016. Program synthesis from polymorphic refinement types. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*. 522–538.

D. Qi, W. N. Sumner, F. Qin, M. Zheng, X. Zhang, and A. Roychoudhury. 2012. Modeling Software Execution Environment. In *2012 19th Working Conference on Reverse Engineering*. 415–424. https://doi.org/10.1109/WCRE.2012.51

Arjun Radhakrishna, Nicholas V. Lewchenko, Shawn Meier, Sergio Mover, Krishna Chaitanya Sripada, Damien Zufferey, Bor-Yuh Evan Chang, and Pavol Černý. 2018. DroidStar: Callback Typestates for Android Classes. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 1160–1170. https://doi.org/10.1145/3180155.3180232

Harald Raffelt, Bernhard Steffen, and Therese Berg. 2005. LearnLib: A Library for Automata Learning and Experimentation. In *Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems* (Lisbon, Portugal) *(FMICS '05)*. ACM, New York, NY, USA, 62–71. https://doi.org/10.1145/1081180.1081189

George Reese. 2000. *Database Programming with JDBC and JAVA*. O'Reilly Media, Inc.

Martin C. Rinard, Jiasi Shen, and Varun Mangalick. 2018. Active Learning for Inference and Regeneration of Computer Programs That Store and Retrieve Data. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Boston, MA, USA) *(Onward! 2018)*. ACM, New York, NY, USA, 12–28. https://doi.org/10.1145/3276954.3276959

Gabriel Rodríguez, José M. Andión, Mahmut T. Kandemir, and Juan Touriño. 2016. Trace-based Affine Reconstruction of Codes. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO '16)*. ACM, New York, NY, USA, 139–149. https://doi.org/10.1145/2854038.2854056

Yukinori Sato, Yasushi Inoguchi, and Tadao Nakamura. 2011. On-the-fly Detection of Precise Loop Nests Across Procedures on a Dynamic Binary Translation System. In *Proceedings of the 8th ACM International Conference on Computing Frontiers* (Ischia, Italy) *(CF '11)*. ACM, New York, NY, USA, Article 25, 10 pages. https://doi.org/10.1145/2016604.2016634

Timos K. Sellis. 1988. Multiple-Query Optimization. *ACM Trans. Database Syst.* 13, 1 (March 1988), 23–52. https://doi.org/10.1145/42201.42203

Burr Settles. 2009. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

Jiasi Shen, Martin Rinard, and Nikos Vasilakis. 2021. Automatic Synthesis of Parallel Unix Commands and Pipelines with KumQuat. *CoRR* abs/2012.15443 (2021). arXiv:2012.15443 https://arxiv.org/abs/2012.15443

Jiasi Shen and Martin C. Rinard. 2019. Using Active Learning to Synthesize Models of Applications That Access Databases. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ,

USA) *(PLDI 2019)*. ACM, New York, NY, USA, 269–285. https://doi.org/10.1145/3314221.3314591

Jiasi Shen and Martin C. Rinard. 2021. Active Learning for Inference and Regeneration of Applications That Access Databases. *ACM Trans. Program. Lang. Syst.* 42, 4, Article 18 (Jan. 2021), 119 pages. https://doi.org/10.1145/3430952

Xujie Si, Woosuk Lee, Richard Zhang, Aws Albarghouthi, Paraschos Koutris, and Mayur Naik. 2018. Syntax-Guided Synthesis of Datalog Programs. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 515–527. https://doi.org/10.1145/3236024.3236034

Calvin Smith and Aws Albarghouthi. 2016. MapReduce Program Synthesis. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. Association for Computing Machinery, New York, NY, USA, 326–340. https://doi.org/10.1145/2908080.2908102

Sunbeom So and Hakjoo Oh. 2017. Synthesizing Imperative Programs from Examples Guided by Static Analysis. In *Static Analysis*, Francesco Ranzato (Ed.). Springer International Publishing, Cham, 364–381.

Sunbeom So and Hakjoo Oh. 2018. Synthesizing Pattern Programs from Examples. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden) *(IJCAI'18)*. AAAI Press, 1618–1624.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. 2006. Combinatorial Sketching for Finite Programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California, USA) *(ASPLOS XII)*. ACM, New York, NY, USA, 404–415. https://doi.org/10.1145/1168857.1168907

Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. 2008. BitBlaze: A New Approach to Computer Security via Binary Analysis. In *Proceedings of the 4th International Conference on Information Systems Security* (Hyderabad, India) *(ICISS '08)*. Springer-Verlag, Berlin, Heidelberg, 1–25. https://doi.org/10.1007/978-3-540-89862-7_1

Saurabh Srivastava, Sumit Gulwani, and Jeffrey S. Foster. 2010. From Program Verification to Program Synthesis. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Madrid, Spain) *(POPL '10)*. Association for Computing Machinery, New York, NY, USA, 313–326. https://doi.org/10.1145/1706299.1706337

J. Tubella and A. Gonzalez. 1998. Control speculation in multithreaded processors through dynamic loop detection. In *Proceedings 1998 Fourth International Symposium on High-Performance Computer Architecture*. 14–23. https://doi.org/10.1109/HPCA.1998.650542

Frits Vaandrager. 2017. Model Learning. *Commun. ACM* 60, 2 (Jan. 2017), 86–95. https://doi.org/10.1145/2967606

Nikos Vasilakis, Achilles Benetopoulos, Shivam Handa, Alizee Schoen, Jiasi Shen, and Martin Rinard. 2021. Supply-Chain Vulnerability Elimination via Active Learning and Regeneration. (2021). ACM Conference on Computer and Communications Security.

Michele Volpato and Jan Tretmans. 2015. Approximate Active Learning of Nondeterministic Input Output Transition Systems. *Electronic Communications of the EASST* 72 (2015).

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017a. Synthesizing Highly Expressive SQL Queries from Input-Output Examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 452–466. https://doi.org/10.1145/3062341.3062365

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017b. Synthesizing Highly Expressive SQL Queries from Input-output Examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. ACM, New York, NY, USA, 452–466. https://doi.org/10.1145/3062341.3062365

Xinyu Wang, Isil Dillig, and Rishabh Singh. 2017c. Program Synthesis Using Abstraction Refinement. *Proc. ACM Program. Lang.* 2, POPL, Article 63 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158151

Xinyu Wang, Isil Dillig, and Rishabh Singh. 2017d. Synthesis of Data Completion Scripts Using Finite Tree Automata. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 62 (Oct. 2017), 26 pages. https://doi.org/10.1145/3133886

Michael Widenius and Davis Axmark. 2002. *Mysql Reference Manual* (1st ed.). O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Jerry Wu. 2018. *Using Dynamic Analysis to Infer Python Programs and Convert Them into Database Programs.* Master's thesis. Massachusetts Institute of Technology. https://hdl.handle.net/1721.1/121643.

Wenfei Wu, Ying Zhang, and Sujata Banerjee. 2016. Automatic Synthesis of NF Models by Program Analysis. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (Atlanta, GA, USA) *(HotNets '16)*. ACM, New York, NY, USA, 29–35. https://doi.org/10.1145/3005745.3005754

Cong Yan, Alvin Cheung, Junwen Yang, and Shan Lu. 2017. Understanding Database Performance Inefficiencies in Real-world Web Applications. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. ACM, New York, NY, USA, 1299–1308.

Junwen Yang, Pranav Subramaniam, Shan Lu, Cong Yan, and Alvin Cheung. 2018. How Not to Structure Your Database-backed Web Applications: A Study of Performance Bugs in the Wild. In *Proceedings of the 40th International Conference*

on Software Engineering (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 800–810. https://doi.org/10.1145/3180155.3180194

Xi Ye, Qiaochu Chen, Xinyu Wang, Isil Dillig, and Greg Durrett. 2020. Sketch-Driven Regular Expression Generation from Natural Language and Examples. *Transactions of the Association for Computational Linguistics* 8 (11 2020), 679–694. https://doi.org/10.1162/tacl_a_00339 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00339/1923648/tacl_a_00339.pdf

A APPENDIX: REGENERATED CODE FOR BENCHMARK APPLICATIONS

A.1 RailsCollab Project Manager Command get_projects_id_messages

```
def get_projects_id_messages (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[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'))
                    outputs.extend(util.get_data(s18, 'messages', 'title'))
                    outputs.extend(util.get_data(s18, 'messages', 'text'))
                    s18_all = s18
                    for s18 in s18_all:
                        s19 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                             users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s18, '
                             messages', 'created_by_id')})
                        outputs.extend(util.get_data(s19, 'users', 'id'))
                        outputs.extend(util.get_data(s19, 'users', 'display_name'))
                        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'))
                        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(s0, 'users', 'id'), 'x1': util.get_one_data(s18, '
                             messages', '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 `
                             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
```

```
43                          else:
44                              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(
                                    s18, 'messages', 'project_id')})
45                              s24 = 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')})
46                          s18 = s18_all
47                          s25 = 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]})
48                          outputs.extend(util.get_data(s25, 'people', 'project_id'))
49                          outputs.extend(util.get_data(s25, 'people', 'user_id'))
50                          s26 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                                messages`.`project_id` = :x0 AND `messages`.`is_important` = 1", {'
                                x0': inputs[1]})
51                          s27 = util.do_sql(conn, "SELECT COUNT(*) FROM `categories`  WHERE `
                                categories`.`project_id` = :x0", {'x0': inputs[1]})
52                          if util.has_rows(s27):
53                              s28 = util.do_sql(conn, "SELECT `categories`.* FROM `categories`
                                     WHERE `categories`.`project_id` = :x0", {'x0': inputs[1]})
54                              outputs.extend(util.get_data(s28, 'categories', 'id'))
55                              outputs.extend(util.get_data(s28, 'categories', 'project_id'))
56                              outputs.extend(util.get_data(s28, 'categories', 'name'))
57                              s28_all = s28
58                              for s28 in s28_all:
59                                  s29 = util.do_sql(conn, "SELECT  `projects`.* FROM `projects`
                                         WHERE `projects`.`id` = :x0 LIMIT 1", {'x0': util.
                                         get_one_data(s28, 'categories', 'project_id')})
60                                  outputs.extend(util.get_data(s29, 'projects', 'id'))
61                                  outputs.extend(util.get_data(s29, 'projects', 'name'))
62                                  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(
                                        s28, 'categories', 'project_id')})
63                                  outputs.extend(util.get_data(s30, 'people', 'project_id'))
64                                  outputs.extend(util.get_data(s30, 'people', 'user_id'))
65                                  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')})
66                                  outputs.extend(util.get_data(s31, 'people', 'project_id'))
67                                  outputs.extend(util.get_data(s31, 'people', 'user_id'))
68                                  if util.has_rows(s31):
69                                      pass
70                                  else:
71                                      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': util.
                                             get_one_data(s28, 'categories', 'project_id')})
72                                      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(s28, 'categories', 'project_id')})
73                                  s28 = s28_all
74                                  pass
75                              else:
76                                  pass
77                      else:
78                          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]})
79                          outputs.extend(util.get_data(s9, 'people', 'project_id'))
80                          outputs.extend(util.get_data(s9, 'people', 'user_id'))
81                          s10 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                                messages`.`project_id` = :x0 AND `messages`.`is_important` = 1", {'
                                x0': inputs[1]})
82                          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'))
                        outputs.extend(util.get_data(s12, 'categories', 'name'))
                        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
                        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[1]})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
        if util.has_rows(s4):
```

```
12              s5 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`  WHERE `companies
                    `.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'company_id')
                    })
13              if util.has_rows(s5):
14                  s6 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`  WHERE `
                        companies`.`id` IS NULL LIMIT 1", {})
15                  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]})
16                  outputs.extend(util.get_data(s7, 'messages', 'id'))
17                  outputs.extend(util.get_data(s7, 'messages', 'project_id'))
18                  outputs.extend(util.get_data(s7, 'messages', 'title'))
19                  outputs.extend(util.get_data(s7, 'messages', 'text'))
20                  if util.has_rows(s7):
21                      s8 = util.do_sql(conn, "SELECT  `projects`.* FROM `projects`  WHERE `
                            projects`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
22                      outputs.extend(util.get_data(s8, 'projects', 'id'))
23                      outputs.extend(util.get_data(s8, 'projects', 'name'))
24                      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]})
25                      outputs.extend(util.get_data(s9, 'people', 'project_id'))
26                      outputs.extend(util.get_data(s9, 'people', 'user_id'))
27                      if util.has_rows(s9):
28                          s11 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                                users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s7, '
                                messages', 'created_by_id')})
29                          outputs.extend(util.get_data(s11, 'users', 'id'))
30                          outputs.extend(util.get_data(s11, 'users', 'display_name'))
31                          s12 = util.do_sql(conn, "SELECT  `milestones`.* FROM `milestones`
                                WHERE `milestones`.`id` = :x0 LIMIT 1", {'x0': util.
                                get_one_data(s7, 'messages', 'milestone_id')})
32                          outputs.extend(util.get_data(s12, 'milestones', 'id'))
33                          outputs.extend(util.get_data(s12, 'milestones', 'project_id'))
34                          outputs.extend(util.get_data(s12, 'milestones', 'name'))
35                          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]})
36                          outputs.extend(util.get_data(s13, 'people', 'project_id'))
37                          outputs.extend(util.get_data(s13, 'people', 'user_id'))
38                          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]})
39                          outputs.extend(util.get_data(s14, 'people', 'project_id'))
40                          outputs.extend(util.get_data(s14, 'people', 'user_id'))
41                          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]})
42                          outputs.extend(util.get_data(s15, 'people', 'project_id'))
43                          outputs.extend(util.get_data(s15, 'people', 'user_id'))
44                          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]})
45                          outputs.extend(util.get_data(s16, 'people', 'project_id'))
46                          outputs.extend(util.get_data(s16, 'people', 'user_id'))
47                          s17 = util.do_sql(conn, "SELECT  `categories`.* FROM `categories`
                                WHERE `categories`.`id` = :x0 LIMIT 1", {'x0': util.
                                get_one_data(s7, 'messages', 'category_id')})
48                          outputs.extend(util.get_data(s17, 'categories', 'id'))
49                          outputs.extend(util.get_data(s17, 'categories', 'name'))
50                          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]})
51                          outputs.extend(util.get_data(s18, 'users', 'id'))
52                          outputs.extend(util.get_data(s18, 'users', 'display_name'))
53                          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]})
54                          outputs.extend(util.get_data(s19, 'people', 'project_id'))
55                          outputs.extend(util.get_data(s19, 'people', 'user_id'))
```

```
                        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]})
                        outputs.extend(util.get_data(s20, 'people', 'project_id'))
                        outputs.extend(util.get_data(s20, 'people', 'user_id'))
                    else:
                        s10 = 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]})
                else:
                    pass
            else:
                pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.3 RailsCollab Project Manager Command get_projects_id_messages_display_list

```
def get_projects_id_messages_display_list (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[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'))
                    outputs.extend(util.get_data(s18, 'messages', 'title'))
                    s18_all = s18
                    for s18 in s18_all:
                        s19 = 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(s19, 'projects', 'id'))
                        outputs.extend(util.get_data(s19, 'projects', 'name'))
```

```
31                      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': util.get_one_data(s18, '
                             messages', 'project_id')})
32                      outputs.extend(util.get_data(s20, 'people', 'project_id'))
33                      outputs.extend(util.get_data(s20, 'people', 'user_id'))
34                      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': util.get_one_data
                             (s18, 'messages', 'project_id')})
35                      outputs.extend(util.get_data(s21, 'people', 'project_id'))
36                      outputs.extend(util.get_data(s21, 'people', 'user_id'))
37                      if util.has_rows(s21):
38                          s25 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                                 users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s18, '
                                 messages', 'created_by_id')})
39                          outputs.extend(util.get_data(s25, 'users', 'id'))
40                          outputs.extend(util.get_data(s25, 'users', 'display_name'))
41                      else:
42                          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': util.get_one_data(
                                 s18, 'messages', 'project_id')})
43                          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': util.
                                 get_one_data(s18, 'messages', 'project_id')})
44                          s24 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                                 users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s18, '
                                 messages', 'created_by_id')})
45                          outputs.extend(util.get_data(s24, 'users', 'id'))
46                          outputs.extend(util.get_data(s24, 'users', 'display_name'))
47                  s18 = s18_all
48                  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]})
49                  outputs.extend(util.get_data(s26, 'people', 'project_id'))
50                  outputs.extend(util.get_data(s26, 'people', 'user_id'))
51                  s27 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                         messages`.`project_id` = :x0 AND `messages`.`is_important` = 1", {'
                         x0': inputs[1]})
52                  s28 = util.do_sql(conn, "SELECT COUNT(*) FROM `categories`  WHERE `
                         categories`.`project_id` = :x0", {'x0': inputs[1]})
53                  if util.has_rows(s28):
54                      s29 = util.do_sql(conn, "SELECT `categories`.* FROM `categories`
                             WHERE `categories`.`project_id` = :x0", {'x0': inputs[1]})
55                      outputs.extend(util.get_data(s29, 'categories', 'id'))
56                      outputs.extend(util.get_data(s29, 'categories', 'project_id'))
57                      outputs.extend(util.get_data(s29, 'categories', 'name'))
58                      s29_all = s29
59                      for s29 in s29_all:
60                          s30 = util.do_sql(conn, "SELECT  `projects`.* FROM `projects`
                                 WHERE `projects`.`id` = :x0 LIMIT 1", {'x0': util.
                                 get_one_data(s29, 'categories', 'project_id')})
61                          outputs.extend(util.get_data(s30, 'projects', 'id'))
62                          outputs.extend(util.get_data(s30, 'projects', 'name'))
63                          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(
                                 s29, 'categories', 'project_id')})
64                          outputs.extend(util.get_data(s31, 'people', 'project_id'))
65                          outputs.extend(util.get_data(s31, 'people', 'user_id'))
66                          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': util.get_one_data(
                                 s29, 'categories', 'project_id')})
67                          outputs.extend(util.get_data(s32, 'people', 'project_id'))
68                          outputs.extend(util.get_data(s32, 'people', 'user_id'))
69                          if util.has_rows(s32):
70                              pass
71                          else:
```

```
72                                  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(s29, 'categories', 'project_id')})
73                                  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(s29, 'categories', 'project_id')})
74                              s29 = s29_all
75                              pass
76                          else:
77                              pass
78                      else:
79                          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]})
80                          outputs.extend(util.get_data(s9, 'people', 'project_id'))
81                          outputs.extend(util.get_data(s9, 'people', 'user_id'))
82                          s10 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                                messages`.`project_id` = :x0 AND `messages`.`is_important` = 1", {'
                                x0': inputs[1]})
83                          s11 = util.do_sql(conn, "SELECT COUNT(*) FROM `categories`  WHERE `
                                categories`.`project_id` = :x0", {'x0': inputs[1]})
84                          if util.has_rows(s11):
85                              s12 = util.do_sql(conn, "SELECT `categories`.* FROM `categories`
                                    WHERE `categories`.`project_id` = :x0", {'x0': inputs[1]})
86                              outputs.extend(util.get_data(s12, 'categories', 'id'))
87                              outputs.extend(util.get_data(s12, 'categories', 'project_id'))
88                              outputs.extend(util.get_data(s12, 'categories', 'name'))
89                              s12_all = s12
90                              for s12 in s12_all:
91                                  s13 = util.do_sql(conn, "SELECT  `projects`.* FROM `projects`
                                        WHERE `projects`.`id` = :x0 LIMIT 1", {'x0': util.
                                        get_one_data(s12, 'categories', 'project_id')})
92                                  outputs.extend(util.get_data(s13, 'projects', 'id'))
93                                  outputs.extend(util.get_data(s13, 'projects', 'name'))
94                                  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')})
95                                  outputs.extend(util.get_data(s14, 'people', 'project_id'))
96                                  outputs.extend(util.get_data(s14, 'people', 'user_id'))
97                                  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')})
98                                  outputs.extend(util.get_data(s15, 'people', 'project_id'))
99                                  outputs.extend(util.get_data(s15, 'people', 'user_id'))
100                                 if util.has_rows(s15):
101                                     pass
102                                 else:
103                                     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')})
104                                     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')})
105                             s12 = s12_all
106                             pass
107                         else:
108                             pass
109             else:
110                 pass
111         else:
112             pass
113     else:
114         pass
115     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`.

```
def get_projects_id_times (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[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 `time_records`  WHERE `
                     time_records`.`project_id` = :x0", {'x0': inputs[1]})
                s8 = util.do_sql(conn, "SELECT COUNT(*) FROM `time_records`  WHERE `
                     time_records`.`project_id` = :x0", {'x0': inputs[1]})
                if util.has_rows(s8):
                    s10 = util.do_sql(conn, "SELECT  `time_records`.* FROM `time_records`
                         WHERE `time_records`.`project_id` = :x0 ORDER BY created_on DESC
                         LIMIT 10 OFFSET 0", {'x0': inputs[1]})
                    outputs.extend(util.get_data(s10, 'time_records', 'id'))
                    outputs.extend(util.get_data(s10, 'time_records', 'project_id'))
                    outputs.extend(util.get_data(s10, 'time_records', 'name'))
                    s10_all = s10
                    for s10 in s10_all:
                        s11 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`
                             WHERE `companies`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data
                             (s10, '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(s10, '
                                 time_records', 'task_id')})
                            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'
                                     ))
                                s32 = 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(s32, 'projects', 'id'))
```

```
37                                  outputs.extend(util.get_data(s32, 'projects', 'name'))
38                                  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(s10, 'time_records', 'project_id')})
39                                  outputs.extend(util.get_data(s33, 'people', 'project_id'))
40                                  outputs.extend(util.get_data(s33, 'people', 'user_id'))
41                                  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')})
42                                  outputs.extend(util.get_data(s34, 'people', 'project_id'))
43                                  outputs.extend(util.get_data(s34, 'people', 'user_id'))
44                                  if util.has_rows(s34):
45                                      pass
46                                  else:
47                                      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')})
48                                      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')})
49                              else:
50                                  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')})
51                                  outputs.extend(util.get_data(s26, 'projects', 'id'))
52                                  outputs.extend(util.get_data(s26, 'projects', 'name'))
53                                  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')})
54                                  outputs.extend(util.get_data(s27, 'people', 'project_id'))
55                                  outputs.extend(util.get_data(s27, 'people', 'user_id'))
56                                  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')})
57                                  outputs.extend(util.get_data(s28, 'people', 'project_id'))
58                                  outputs.extend(util.get_data(s28, 'people', 'user_id'))
59                                  if util.has_rows(s28):
60                                      pass
61                                  else:
62                                      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')})
63                                      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')})
64                          else:
65                              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')})
66                              outputs.extend(util.get_data(s12, 'users', 'display_name'))
67                              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')})
68                              if util.has_rows(s13):
69                                  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(s0, '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 `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(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')})
                                    s24 = 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:
                                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 `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(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')})
                                    s18 = 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')})
                        s10 = s10_all
                        s37 = 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(s37, 'people', 'project_id'))
                        outputs.extend(util.get_data(s37, 'people', 'user_id'))
```

```
                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'))
            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(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'))
        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  `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  `task_lists`.* FROM `
                            task_lists`  WHERE `task_lists`.`id` = :x0 LIMIT 1", {'
                            x0': util.get_one_data(s47, 'tasks', 'task_list_id')})
                        outputs.extend(util.get_data(s51, 'task_lists', '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'))
                            s53 = 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(s53, 'people', 'project_id'
                                ))
                            outputs.extend(util.get_data(s53, 'people', 'user_id'))
                            s54 = 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(s54, 'people', 'project_id'
                                ))
                            outputs.extend(util.get_data(s54, 'people', 'user_id'))
                        else:
                            pass
                    else:
                        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'))
                        outputs.extend(util.get_data(s48, 'people', 'user_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'))
                        outputs.extend(util.get_data(s49, 'people', 'user_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'))
                        outputs.extend(util.get_data(s50, 'people', 'user_id'))
                else:
                    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', 'display_name'))
                    s39 = 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(s39, 'tasks', 'id'))
                    outputs.extend(util.get_data(s39, 'tasks', 'text'))
                    if util.has_rows(s39):
                        s43 = util.do_sql(conn, "SELECT  `task_lists`.* FROM `
                            task_lists`  WHERE `task_lists`.`id` = :x0 LIMIT 1", {'
                            x0': util.get_one_data(s39, 'tasks', 'task_list_id')})
                        outputs.extend(util.get_data(s43, 'task_lists', '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]})
```

```
70                                  outputs.extend(util.get_data(s44, 'people', 'project_id'
                                        ))
71                                  outputs.extend(util.get_data(s44, 'people', 'user_id'))
72                                  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]})
73                                  outputs.extend(util.get_data(s45, 'people', 'project_id'
                                        ))
74                                  outputs.extend(util.get_data(s45, 'people', 'user_id'))
75                                  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]})
76                                  outputs.extend(util.get_data(s46, 'people', 'project_id'
                                        ))
77                                  outputs.extend(util.get_data(s46, 'people', 'user_id'))
78                              else:
79                                  pass
80                          else:
81                              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]})
82                              outputs.extend(util.get_data(s40, 'people', 'project_id'))
83                              outputs.extend(util.get_data(s40, 'people', 'user_id'))
84                              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]})
85                              outputs.extend(util.get_data(s41, 'people', 'project_id'))
86                              outputs.extend(util.get_data(s41, 'people', 'user_id'))
87                              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]})
88                              outputs.extend(util.get_data(s42, 'people', 'project_id'))
89                              outputs.extend(util.get_data(s42, 'people', 'user_id'))
90                      else:
91                          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]})
92                          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')})
93                          outputs.extend(util.get_data(s11, 'companies', 'name'))
94                          if util.has_rows(s11):
95                              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')})
96                              outputs.extend(util.get_data(s25, 'tasks', 'id'))
97                              outputs.extend(util.get_data(s25, 'tasks', 'text'))
98                              if util.has_rows(s25):
99                                  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')})
100                                 outputs.extend(util.get_data(s31, 'task_lists', 'project_id'
                                        ))
101                                 if util.has_rows(s31):
102                                     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]})
103                                     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]})
104                                     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]})
```

```
105                             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]})
106                             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]})
107                         else:
108                             pass
109                     else:
110                         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]})
111                         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]})
112                         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]})
113                         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]})
114                         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]})
115                 else:
116                     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')})
117                     outputs.extend(util.get_data(s12, 'users', 'display_name'))
118                     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')})
119                     outputs.extend(util.get_data(s13, 'tasks', 'id'))
120                     outputs.extend(util.get_data(s13, 'tasks', 'text'))
121                     if util.has_rows(s13):
122                         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')})
123                         outputs.extend(util.get_data(s19, 'task_lists', 'project_id'
                                 ))
124                         if util.has_rows(s19):
125                             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]})
126                             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]})
127                             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]})
128                             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]})
129                             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]})
130                         else:
131                             pass
132                     else:
```

```
                                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]})
                                s15 = 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]})
                                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]})
                                s17 = 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]})
                                s18 = 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:
                pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.6 RailsCollab Project Manager Command `get_projects_id_milestones_id`

```
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  `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'))
        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  `milestones`.* FROM `milestones`  WHERE `
                     milestones`.`project_id` = :x0 AND `milestones`.`id` = :x1 LIMIT 1", {'
                     x0': inputs[1], 'x1': inputs[2]})
                outputs.extend(util.get_data(s7, 'milestones', 'id'))
                outputs.extend(util.get_data(s7, 'milestones', 'project_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`.* 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'))
                    s10 = 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]})
                    outputs.extend(util.get_data(s10, 'people', 'project_id'))
                    outputs.extend(util.get_data(s10, 'people', 'user_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'))
                            outputs.extend(util.get_data(s19, 'messages', 'title'))
                            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(s19, '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(s0, '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(s0, 'users', 'id'), 'x1': inputs[1]})
                            outputs.extend(util.get_data(s23, 'people', 'project_id'))
                            outputs.extend(util.get_data(s23, 'people', 'user_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(s0, 'users', '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'))
                            outputs.extend(util.get_data(s13, 'messages', 'title'))
                            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]})
```

```
                            outputs.extend(util.get_data(s15, 'task_lists', 'id'))
                            outputs.extend(util.get_data(s15, 'task_lists', 'milestone_id'))
                            outputs.extend(util.get_data(s15, 'task_lists', 'name'))
                            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  `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(s17, 'people', 'project_id'))
                            outputs.extend(util.get_data(s17, 'people', 'user_id'))
                            s18 = 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(s18, 'people', 'project_id'))
                            outputs.extend(util.get_data(s18, 'people', 'user_id'))
                    else:
                        pass
                else:
                    pass
            else:
                pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

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': inputs[0]})
    outputs.extend(util.get_data(s0, 'users', 'id'))
    outputs.extend(util.get_data(s0, 'users', 'company_id'))
    outputs.extend(util.get_data(s0, 'users', 'display_name'))
    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', 'company_id'))
        outputs.extend(util.get_data(s1, 'users', 'display_name'))
        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`", {})
        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", {})
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.8 RailsCollab Project Manager Command get_companies_id

```
def get_companies_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  `companies`.* FROM `companies`  WHERE `companies`.`
             id` = :x0 LIMIT 1", {'x0': inputs[1]})
        outputs.extend(util.get_data(s4, 'companies', 'id'))
        outputs.extend(util.get_data(s4, 'companies', 'name'))
        outputs.extend(util.get_data(s4, 'companies', 'email'))
        outputs.extend(util.get_data(s4, 'companies', 'homepage'))
        outputs.extend(util.get_data(s4, 'companies', 'address'))
        outputs.extend(util.get_data(s4, 'companies', 'address2'))
        outputs.extend(util.get_data(s4, 'companies', 'city'))
        outputs.extend(util.get_data(s4, 'companies', 'state'))
        outputs.extend(util.get_data(s4, 'companies', 'zipcode'))
        outputs.extend(util.get_data(s4, 'companies', 'country'))
        outputs.extend(util.get_data(s4, 'companies', 'phone_number'))
        outputs.extend(util.get_data(s4, 'companies', 'fax_number'))
        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 %>" /></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.er

```
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]})
    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  `users`.* FROM `users`  WHERE `users`.`id` = :x0
             LIMIT 1", {'x0': inputs[1]})
        outputs.extend(util.get_data(s4, 'users', 'id'))
        outputs.extend(util.get_data(s4, 'users', 'company_id'))
        outputs.extend(util.get_data(s4, 'users', 'email'))
```

```
        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', 'mobile_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", {'x0': inputs[1]})
                    if util.has_rows(s9):
                        s10 = util.do_sql(conn, "SELECT `user_im_values`.* 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(s10, 'user_im_values', 'user_id'))
                        outputs.extend(util.get_data(s10, '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', 'name'))
                            outputs.extend(util.get_data(s11, 'im_types', 'icon'))
                        s10 = s10_all
                        pass
                    else:
                        pass
                else:
                    pass
            else:
                pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.10 Kanban Task Manager Command `get_api_lists`

```
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'))
            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', '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'))
                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
                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', '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', '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'))
                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
                pass
            s3 = s3_all
            pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.12 Kanban Task Manager Command get_api_cards

```
def get_api_cards (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 `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", {'x0': util.get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s1, 'cards', 'id'))
        outputs.extend(util.get_data(s1, 'cards', 'list_id'))
        outputs.extend(util.get_data(s1, 'cards', 'title'))
        outputs.extend(util.get_data(s1, 'cards', 'description'))
        outputs.extend(util.get_data(s1, 'cards', 'due_date'))
        outputs.extend(util.get_data(s1, 'cards', 'position'))
        s1_all = s1
        for s1 in s1_all:
            s2 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `
                 card_comments`.`card_id` = :x0", {'x0': util.get_one_data(s1, 'cards', 'id'
                 )})
            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'))
            outputs.extend(util.get_data(s4, 'users', 'email'))
            outputs.extend(util.get_data(s4, 'users', 'bio'))
            outputs.extend(util.get_data(s4, 'users', 'full_name'))
            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')})
            outputs.extend(util.get_data(s5, 'card_comments', 'card_id'))
            outputs.extend(util.get_data(s5, 'card_comments', 'content'))
            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', 'card_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'))
                outputs.extend(util.get_data(s6, 'cards', 'position'))
                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'))
                outputs.extend(util.get_data(s7, 'users', 'full_name'))
            s5 = s5_all
            pass
        s1 = s1_all
        pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.13 Kanban Task Manager Command get_api_cards_id

```
def get_api_cards_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 `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 AND `cards`.`id` = :x1  ORDER BY `
            cards`.`position` ASC, `lists`.`position` ASC LIMIT 1", {'x0': util.
            get_one_data(s0, 'users', 'id'), 'x1': inputs[1]})
        outputs.extend(util.get_data(s1, 'cards', 'id'))
        outputs.extend(util.get_data(s1, 'cards', 'list_id'))
        outputs.extend(util.get_data(s1, 'cards', 'title'))
        outputs.extend(util.get_data(s1, 'cards', 'description'))
        outputs.extend(util.get_data(s1, 'cards', 'due_date'))
        outputs.extend(util.get_data(s1, 'cards', 'position'))
        if util.has_rows(s1):
            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'))
            outputs.extend(util.get_data(s4, 'users', 'email'))
            outputs.extend(util.get_data(s4, 'users', 'bio'))
            outputs.extend(util.get_data(s4, 'users', 'full_name'))
            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'))
            outputs.extend(util.get_data(s5, 'card_comments', 'content'))
            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', 'card_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'))
                outputs.extend(util.get_data(s6, 'cards', 'position'))
                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'))
                outputs.extend(util.get_data(s7, 'users', 'full_name'))
            s5 = s5_all
            pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.14 Kanban Task Manager Command get_api_boards_id

```
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  `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': util.get_one_data(s0, 'users',
             'id'), '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', 'board_id'))
            outputs.extend(util.get_data(s2, 'lists', 'title'))
            outputs.extend(util.get_data(s2, 'lists', 'position'))
            s2_all = s2
            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,
                      'lists', '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', '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'))
                    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
                    pass
                s3 = s3_all
                pass
            s2 = s2_all
            s9 = util.do_sql(conn, "SELECT `users`.* FROM `users` INNER JOIN `board_members`
                 ON `users`.`id` = `board_members`.`member_id` WHERE `board_members`.`
                board_id` = :x0  ORDER BY `users`.`id` ASC", {'x0': inputs[1]})
            outputs.extend(util.get_data(s9, 'users', 'id'))
            outputs.extend(util.get_data(s9, 'users', 'email'))
            outputs.extend(util.get_data(s9, 'users', 'bio'))
```

```
            outputs.extend(util.get_data(s9, 'users', 'full_name'))
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.15 Kanban Task Manager Command get_api_users_current

```
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'))
    outputs.extend(util.get_data(s0, 'users', 'full_name'))
    return util.add_warnings(outputs)
```

A.16 Kanban Task Manager Command get_api_users_id

```
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'))
    outputs.extend(util.get_data(s0, 'users', 'full_name'))
    return util.add_warnings(outputs)
```

A.17 Todo Task Manager Command get_home

```
def get_home (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT `lists`.* FROM `lists`", {})
    outputs.extend(util.get_data(s0, 'lists', 'id'))
    outputs.extend(util.get_data(s0, 'lists', 'name'))
    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'))
            outputs.extend(util.get_data(s3, 'tasks', 'list_id'))
            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
    pass
    return util.add_warnings(outputs)
```

A.18 Todo Task Manager Command get_lists_id_tasks

```
def get_lists_id_tasks (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT `lists`.* FROM `lists`", {})
    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')})
            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
    s5 = util.do_sql(conn, "SELECT  `lists`.* FROM `lists` WHERE `lists`.`id` = :x0 LIMIT 1"
         , {'x0': inputs[0]})
    outputs.extend(util.get_data(s5, 'lists', 'id'))
    outputs.extend(util.get_data(s5, 'lists', 'name'))
    return util.add_warnings(outputs)
```

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`.

```
def get_lists_id_tasks (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT `lists`.* FROM `lists`", {})
    outputs.extend(util.get_data(s0, 'lists', 'id'))
    outputs.extend(util.get_data(s0, 'lists', 'name'))
    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'))
            outputs.extend(util.get_data(s3, 'tasks', 'list_id'))
            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
    s5 = util.do_sql(conn, "SELECT  `lists`.* FROM `lists` WHERE `lists`.`id` = :x0 LIMIT 1"
         , {'x0': inputs[0]})
    return util.add_warnings(outputs)
```

A.20 Fulcrum Task Manager Command `get_home`

```
def get_home (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):
```

```
7         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')
              })
8         outputs.extend(util.get_data(s1, 'users', 'email'))
9         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')})
10        outputs.extend(util.get_data(s2, 'projects', 'id'))
11        outputs.extend(util.get_data(s2, 'projects', 'name'))
12        outputs.extend(util.get_data(s2, 'projects', 'start_date'))
13        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')
              })
14        outputs.extend(util.get_data(s3, 'users', 'email'))
15        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", {'x0': util.get_one_data(s0, 'users', 'id')})
16        outputs.extend(util.get_data(s4, 'projects', 'id'))
17        outputs.extend(util.get_data(s4, 'projects', 'name'))
18        outputs.extend(util.get_data(s4, 'projects', 'start_date'))
19    else:
20        pass
21    return util.add_warnings(outputs)
```

A.21 Fulcrum Task Manager Command get_projects

```
1  def get_projects (conn, inputs):
2      util.clear_warnings()
3      outputs = []
4      s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`email` = :x0
           ORDER BY `users`.`id` ASC LIMIT 1", {'x0': inputs[0]})
5      outputs.extend(util.get_data(s0, 'users', 'email'))
6      if util.has_rows(s0):
7          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')
               })
8          outputs.extend(util.get_data(s1, 'users', 'email'))
9          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')})
10         outputs.extend(util.get_data(s2, 'projects', 'id'))
11         outputs.extend(util.get_data(s2, 'projects', 'name'))
12         outputs.extend(util.get_data(s2, 'projects', 'start_date'))
13         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')
               })
14         outputs.extend(util.get_data(s3, 'users', 'email'))
15         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", {'x0': util.get_one_data(s0, 'users', 'id')})
16         outputs.extend(util.get_data(s4, 'projects', 'id'))
17         outputs.extend(util.get_data(s4, 'projects', 'name'))
18         outputs.extend(util.get_data(s4, 'projects', 'start_date'))
19     else:
20         pass
21     return util.add_warnings(outputs)
```

A.22 Fulcrum Task Manager Command get_projects_id

```
1  def get_projects_id (conn, inputs):
2      util.clear_warnings()
3      outputs = []
4      s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`email` = :x0
           ORDER BY `users`.`id` ASC LIMIT 1", {'x0': inputs[0]})
5      if util.has_rows(s0):
6          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]})
        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 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` 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`.* 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.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`.* 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):
            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')})
```

```
21                  outputs.extend(util.get_data(s7, 'notes', 'id'))
22                  outputs.extend(util.get_data(s7, 'notes', 'note'))
23                  outputs.extend(util.get_data(s7, 'notes', 'user_id'))
24                  outputs.extend(util.get_data(s7, 'notes', 'story_id'))
25              else:
26                  pass
27          else:
28              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
                     ')})
29      else:
30          pass
31      return util.add_warnings(outputs)
```

A.24 Fulcrum Task Manager Command get_projects_id_stories_id

```
1   def get_projects_id_stories_id (conn, inputs):
2       util.clear_warnings()
3       outputs = []
4       s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`email` = :x0
             ORDER BY `users`.`id` ASC LIMIT 1", {'x0': inputs[0]})
5       if util.has_rows(s0):
6           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')
                 })
7           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')})
8           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')
                 })
9           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]})
10          if util.has_rows(s4):
11              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]})
12              outputs.extend(util.get_data(s6, 'stories', 'id'))
13              outputs.extend(util.get_data(s6, 'stories', 'title'))
14              outputs.extend(util.get_data(s6, 'stories', 'description'))
15              outputs.extend(util.get_data(s6, 'stories', 'estimate'))
16              outputs.extend(util.get_data(s6, 'stories', 'requested_by_id'))
17              outputs.extend(util.get_data(s6, 'stories', 'owned_by_id'))
18              outputs.extend(util.get_data(s6, 'stories', 'project_id'))
19              if util.has_rows(s6):
20                  s8 = util.do_sql(conn, "SELECT `notes`.* FROM `notes`  WHERE `notes`.`
                         story_id` = :x0", {'x0': inputs[2]})
21                  outputs.extend(util.get_data(s8, 'notes', 'id'))
22                  outputs.extend(util.get_data(s8, 'notes', 'note'))
23                  outputs.extend(util.get_data(s8, 'notes', 'user_id'))
24                  outputs.extend(util.get_data(s8, 'notes', 'story_id'))
25              else:
26                  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(s0,
                          'users', 'id')})
27          else:
28              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
                     ')})
29      else:
30          pass
31      return util.add_warnings(outputs)
```

A.25 Fulcrum Task Manager Command get_projects_id_stories_id_notes

```
def get_projects_id_stories_id_notes (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):
            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):
                s8 = util.do_sql(conn, "SELECT `notes`.* 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', 'note'))
                outputs.extend(util.get_data(s8, 'notes', 'user_id'))
                outputs.extend(util.get_data(s8, 'notes', 'story_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(s0,
                      'users', 'id')})
        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.26 Fulcrum Task Manager Command get_projects_id_stories_id_notes_id

```
def get_projects_id_stories_id_notes_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):
```

```
            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):
                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'))
                outputs.extend(util.get_data(s8, 'notes', 'note'))
                outputs.extend(util.get_data(s8, 'notes', 'user_id'))
                outputs.extend(util.get_data(s8, 'notes', 'story_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(s0, '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(s0,
                      'users', 'id')})
        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.27 Fulcrum Task Manager 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`.`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]})
        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 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` 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`.* 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 `
                 activities`.`channel_id` IN :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')})
                s11 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`id`
                     = :x0 LIMIT 1", {'x0': util.get_one_data(s0, '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'))
                        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
                s12 = s12_all
                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')})
                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:
```

```
                    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(s2):
            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):
                s10 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.`id` IN :x0", {'x0': util.get_data(s5, 'activities', 'user_id')})
                s11 = util.do_sql(conn, "SELECT `users`.* FROM `users` WHERE `users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
                s12 = util.do_sql(conn, "SELECT `channels`.* FROM `channels` WHERE `channels`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
                if util.has_rows(s12):
                    s13 = 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'))
                    outputs.extend(util.get_data(s13, 'activities', 'user_id'))
                    s14 = 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'))
                    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'))
                        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')})
```

```
                s7 = util.do_sql(conn, "SELECT  `channels`.* FROM `channels`  WHERE `
                    channels`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
                if util.has_rows(s7):
                    s8 = util.do_sql(conn, "SELECT  `activities`.* FROM `activities`  WHERE
                        `activities`.`channel_id` = :x0 ORDER BY id LIMIT 1", {'x0': inputs
                        [1]})
                    s9 = util.do_sql(conn, "SELECT  `activities`.* FROM `activities`  WHERE
                        `activities`.`channel_id` = :x0 ORDER BY id DESC LIMIT 30", {'x0':
                        inputs[1]})
                else:
                    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`  WHERE `channels`.`
                id` = :x0 LIMIT 1", {'x0': inputs[1]})
    else:
        pass
    return util.add_warnings(outputs)
```

A.30 Kandan Chat Room Command get_channels_id_activities_id

```
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': 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 `
                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(s5, 'activities', 'user_id')})
                s9 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`id` =
                     :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
                s10 = util.do_sql(conn, "SELECT  `activities`.* FROM `activities`  WHERE `
                    activities`.`id` = :x0 LIMIT 1", {'x0': inputs[2]})
                outputs.extend(util.get_data(s10, 'activities', 'content'))
            else:
                s6 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`id` =
                     :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
                s7 = util.do_sql(conn, "SELECT  `activities`.* 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`.* FROM `users`  WHERE `users`.`id` = :x0
                 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
            s4 = util.do_sql(conn, "SELECT  `activities`.* FROM `activities`  WHERE `
                activities`.`id` = :x0 LIMIT 1", {'x0': inputs[2]})
            outputs.extend(util.get_data(s4, 'activities', 'content'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.31 Kandan Chat Room Command get_me

```
def get_me (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'))
    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_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.32 Kandan Chat Room Command get_users

```
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[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):
```

```
            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(s5, 'activities', 'user_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'))
                outputs.extend(util.get_data(s8, 'users', 'username'))
                s9 = 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(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'))
                outputs.extend(util.get_data(s9, 'users', 'username'))
                s10 = util.do_sql(conn, "SELECT `users`.* FROM `users`", {})
                outputs.extend(util.get_data(s10, 'users', 'id'))
                outputs.extend(util.get_data(s10, 'users', 'email'))
                outputs.extend(util.get_data(s10, 'users', 'first_name'))
                outputs.extend(util.get_data(s10, 'users', 'last_name'))
                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(s0, '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'))
                outputs.extend(util.get_data(s6, 'users', 'username'))
                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(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'))
            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'))
            outputs.extend(util.get_data(s4, 'users', 'username'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.33 Kandan Chat Room Command get_users_id

```
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'))
    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_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`.`tag_id` WHERE `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`.`tag_id` WHERE `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'))
    outputs.extend(util.get_data(s0, 'comments', 'body'))
    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", {})
```

```
        outputs.extend(util.get_data(s1, 'pages', 'id'))
        outputs.extend(util.get_data(s1, 'pages', 'title'))
        outputs.extend(util.get_data(s1, 'pages', 'slug'))
        outputs.extend(util.get_data(s1, 'pages', 'body'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.38 Enki Blogging Application Command get_admin_pages_id

```
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

```
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  `posts`.* 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 COUNT(*) FROM `comments` WHERE `comments`.`
                 post_id` = :x0", {'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.

```
def get_admin (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT  posts.*, max(comments.created_at), comments.post_id FROM
          `posts` INNER JOIN comments ON comments.post_id = posts.id GROUP BY comments.
         post_id, posts.id, posts.title, posts.slug, posts.body, posts.body_html, posts.
         active, posts.approved_comments_count, posts.cached_tag_list, posts.published_at,
         posts.created_at, posts.updated_at, posts.edited_at  ORDER BY max(comments.
         created_at) desc LIMIT 5", {})
    s0_all = s0
    for s0 in s0_all:
        s1 = util.do_sql(conn, "SELECT  `comments`.* 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', 'title'))
    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')})
    s5 = s5_all
    pass
    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`", {})
    outputs.extend(util.get_data(s0, 'articles', 'id'))
    outputs.extend(util.get_data(s0, 'articles', 'title'))
    outputs.extend(util.get_data(s0, 'articles', 'text'))
    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'))
        outputs.extend(util.get_data(s1, 'comments', 'article_id'))
    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
```

```
            pass
        else:
            pass
    else:
        pass
    return util.add_warnings(outputs)
```

A.44 Synthetic Program repeat_2

```
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

```
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

```
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

```
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

```
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 s1 in s1_all:
            s2 = util.do_sql(conn, "SELECT * FROM t2", {})
        s1 = s1_all
        pass
    s0 = s0_all
    pass
    return util.add_warnings(outputs)
```

A.49 Synthetic Program after_2

```
def after_2 (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
    pass
    return util.add_warnings(outputs)
```

A.50 Synthetic Program after_3

```
def after_3 (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
    pass
    return util.add_warnings(outputs)
```

A.51 Synthetic Program after_4

```
def after_4 (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
    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)
```