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 looping structure 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 looping structures otherwise undetectable given only the ability to observe application behavior. Active loop detection therefore enables Shear to infer a broader range of looping structures than previous approaches.

1 INTRODUCTION

The need to infer loop constructs from observations of program executions has repeatedly arisen in a range of fields [Biermann et al. 1975; Burtscher et al. 2005; Caballero et al. 2009; Elnozahy 1999; Heule et al. 2015; Ketterlin and Clauss 2008; Kobayashi 1984; Noughi et al. 2014; Qi et al. 2012; Rodríguez et al. 2016; Shen and Rinard 2019; Wu 2018]. Faced with this problem, researchers have developed a range of approaches that attempt to recognize repetitive structures in these observations and infer the underlying presence of loops from these repetitive structures. One 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]. 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 deploy algorithms that work for specific cases such as linear sequences of accessed addresses [Burtscher et al. 2005; Elnozahy 1999; Ketterlin and Clauss 2008; Rodríguez et al. 2016]. The limitations present in each of these techniques can constrain their applicability by limiting the range of loops that they can effectively infer.

We present a new loop inference algorithm. In contrast to previous loop inference algorithms, which simply observe program executions, 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 executed 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. In this context loop inference is important because many such programs contain loops that iterate over the results of database queries. Shear works with applications that conform to the Shear DSL, which has loops that iterate over collections of rows returned from database queries (we note that loops that iterate over collections arise in many domains [Caballero et al. 2009; Caballero and Song 2013; Dean and Ghemawat 2008; Heule et al. 2015; Liskov et al. 1977; Long et al. 2014; Noughi et al. 2014; Shen and Rinard 2019; Wu 2018; Zaharia et al. 2012]).

Here Shear manipulates the traffic between the program and the database to disambiguate looping and repetitive constructs that operate on rows returned from database queries. We compare the

.

Shear loop inference algorithm to the heuristic loop inference algorithm present in Konure [Shen and Rinard 2019], a previous system for inferring programs that access relational databases, as well as other previous loop detection systems [Kobayashi 1984; Noughi et al. 2014]. 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 4).

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 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. From the inferred program, Shear regenerates new implementations that can contain optimizations, cleaner code, and systematically added security checks. Like Konure, Shear also infers the program as a black box, without requiring access to the source code or the binary of the program. Hence Shear works well with programs written in any language or implementation styles (including Ruby on Rails, Java, Python, and many other potential frameworks), as long as the externally visible behavior of the programs can be expressed in the Shear DSL.
- **Results:** It presents experimental results from our Shear implementation. We used it to infer and regenerate a number of applications in Java, Python, and Ruby on Rails, some of which violate the restrictions in Konure. These results highlight the effectiveness of our active loop detection approach in eliminating many of the ad-hoc limitations present in the Konure DSL.
- **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, program comprehension, and migration. The paper discusses how to apply active loop detection in other contexts, specifically as potential extensions to the Mimic [Heule et al. 2015], DaViS [Noughi et al. 2014], Nero [Wu 2018], and Dispatcher [Caballero et al. 2009; Caballero and Song 2013] systems. These potential extensions highlight the generalizability of active loop detection.

The remainder of the paper is structured as follows. Section 2 presents an example that demonstrates active loop detection in Shear. Section 3 presents the design of Shear. Section 4 presents the experimental results evaluating our Shear implementation. Section 5 discusses ways to incorporate active loop detection into other systems. Section 6 presents related work.

2 EXAMPLE

We present an example that illustrates how Shear identifies loops in a database-backed program by manipulating the program's interactions with the environment. The example program in Figure 1 takes an input argument, `tid`. The program retrieves data from three tables: tasks, comments, and users. It first retrieves a task specified by the input. When the task exists, the program retrieves the comments under this task. Then, for each comment, the program retrieves the user that made this comment, along with all of the tasks created by this user. After iterating over the comments, the

```
1  tasks1 = do_sql("SELECT * FROM tasks WHERE id = :x", {"x": tid})
2  output(tasks1, 'title')
3  if has_rows(tasks1):
4    comments = do_sql("SELECT * FROM comments WHERE task_id = :x", {"x": tid})
5    output(comments, 'content')
6    for c in comments:
7      cid = get_value(c, 'commenter_id')
8      users1 = do_sql("SELECT * FROM users WHERE id = :x", {"x": cid})
9      output(users1, 'name')
10     tasks2 = do_sql("SELECT * FROM tasks WHERE creator_id = :x", {"x": cid})
11     output(tasks2, 'title')
12   aid = get_value(tasks1, 'assignee_id')
13   users2 = do_sql("SELECT * FROM users WHERE id = :x", {"x": aid})
14   output(users2, 'name')
15   tasks3 = do_sql("SELECT * FROM tasks WHERE creator_id = :x", {"x": aid})
16   output(tasks3, 'title')
```

Fig. 1. Example program

```
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
```

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

(b) Table tasks

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

(c) Table comments

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

Fig. 2. Example execution

program retrieves the user to which the task is assigned. As with commenters, the program also retrieves all of the tasks created by the assignee user.

The example trace in Figure 2a is obtained from executing the program with the tasks table in Figure 2b, the comments table in Figure 2c, the users table empty, and the input `tid=2`. The trace may be produced by five plausible loop structures:

- **Plausible Loop Structure L (Correct):** A program first performs queries q_0 and q_1. Then, a loop iterates over the two rows retrieved by query q_1. The first iteration performs queries q_2 and q_3. The second iteration performs queries q_4 and q_5. After the loop ends, the program performs queries q_6 and q_7.
- **Plausible Loop Structure W (Incorrect):** A program first performs queries q_0 and q_1. Then, a loop iterates over the two rows retrieved by query q_1. The first iteration performs queries q_2 and q_3. The second iteration performs queries q_4, q_5, and q_6. This loop body contains a conditional statement on the second query in the loop body. When this query retrieves nonempty rows (such as q_3), the loop iteration ends. When this query retrieves empty (such as q_5), the loop iteration performs one more query (q_6). After the loop ends, the program performs query q_7.
- **Plausible Loop Structure X (Incorrect):** A program first performs queries q_0 and q_1. Then, a loop iterates over the two rows retrieved by query q_1. The first iteration performs queries q_2 and q_3. The second iteration performs queries q_4, q_5, q_6, and q_7. This loop body contains

a conditional statement on the second query in the loop body. When this query retrieves nonempty rows (such as q_3), the loop iteration ends. When this query retrieves empty (such as q_5), the loop iteration performs two more queries (q_6 and q_7).

- **Plausible Loop Structure Y (Incorrect):** A program first performs queries q_0 and q_1. Then, a loop iterates over the two rows retrieved by query q_1. The first iteration performs queries q_2, q_3, q_4, and q_5. The second iteration performs queries q_6 and q_7. This loop body contains a conditional statement on the second query in the loop body. When this query retrieves nonempty rows (such as q_3), the loop iteration performs two more queries (q_4 and q_5). When this query retrieves empty (such as q_7), the loop iteration ends.
- **Plausible Loop Structure Z (Incorrect):** A program first performs queries q_0, q_1, q_2, and q_3. Then, a loop iterates over the two rows retrieved by query q_3. The first iteration performs q_4 and q_5. The second iteration performs q_6 and q_7.

Instead of using heuristics to resolve these different possibilities, Shear uses *active loop detection* to infer a unique correct loop structure that produces this trace. 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 potential loop 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.

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 the SQL query into q_1'', which retrieves only the second row among the rows that would have been retrieved by query q_1. The database performs the query 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.

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.

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 the queries q_0, q_1', q_1'', and q_1''' are produced before any iterations of the hypothetical loop start.

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.

Shear then uses this number to identify the queries that would be produced by the hypothetical loop iterations. Specifically, Shear removes the last 2 queries in each altered trace and removes the leading queries up to the hypothetical loop location. The remaining queries in the first altered

trace are queries q_2 and q_3, which would be produced by the first hypothetical loop iteration. The remaining queries in the second altered trace are queries q_4 and q_5, which would be produced by the second hypothetical loop iteration. The remaining queries in the third altered trace are queries q_2, q_3, q_4, and q_5, which would be produced by both of the 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 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 Structure 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.

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.

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 Structure Z.

Identifying Loop 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 query q_1, SHEAR obtains an altered trace where the query q_1 is altered to retrieve only that single row. In this example, because query q_1 retrieves only two rows, these altered traces are already obtained earlier when SHEAR detects the existence of the loop. SHEAR compares all of these altered 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 then used to identify the number of queries that are 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 Structure W,X,Y and determines that the Plausible Loop Structure L is correct.

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 looping and

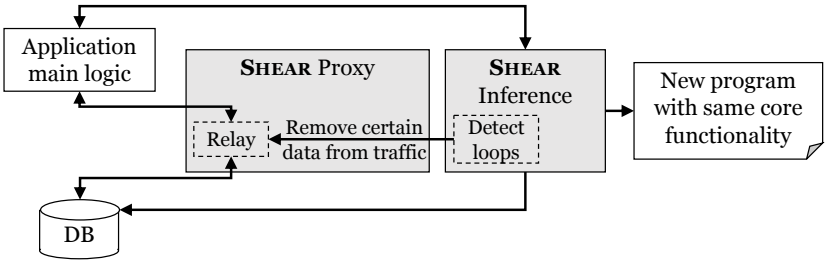

Fig. 3. The SHEAR architecture, including a proxy between the application and the database, which observes and manipulates the database traffic.

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

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

Fig. 4. Grammar for the SHEAR DSL

repetition structures including nested loops, consecutive loops, loops with conditional statements, and non-loop repetitive queries.

3 DESIGN

SHEAR takes a database-backed program, infers its functionality, then regenerates a new version of the program with the same inferred functionality (Figure 3). A key idea of SHEAR is to alter the program's interaction with the environment at a precisely chosen execution points. With a small number of such altered executions, SHEAR learns if the execution point contains a loop. SHEAR represents the detected loops as *loop layout trees* (Section 3.2), which guide the exploration of various paths in the program.

Because SHEAR manipulates program executions on the fly, it is capable of precisely identifying looping structures in the execution traces even when the program may contain spurious repetitions, nested loops, and after-loop subprograms (Section 3.3). We present the active loop detection algorithm and show that it is guaranteed to be sound for programs in the SHEAR DSL (Section 3.5).

3.1 SHEAR Domain-Specific Language

SHEAR works with programs whose functionality can be expressed in the SHEAR DSL. Figure 4 presents the (abstract) grammar for the SHEAR DSL. A program consists of a sequence of Query statements potentially terminated by an If statement. An If statement tests if the Query in the condition retrieves empty or nonempty data. A For statement iterates over the rows in its Query. Each query performs an SQL `select` operation that retrieves data from specified columns in specified tables. 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

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

Fig. 5. SHEAR contexts

$$
\begin{aligned}
\text{Tree} \quad &:= \quad \text{Nil} \mid (Q, r) \setminus \text{Tree} \mid (Q, r) \circlearrowleft \text{Tree}_1 \ldots \text{Tree}_r \setminus \text{Tree}_0 \\
& Q \in \text{Query}, \quad r \in \mathbb{Z}_{\geq 0}
\end{aligned}
$$

Fig. 6. Grammar for loop layout trees

the retrieved data in a unique variable (y) for later use. All variables must be defined before they are used. 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.

Multiple For loops may be nested. To enable the SHEAR inference algorithm to effectively distinguish If statements from Seq statements, SHEAR requires the two branches of each If statement to start with queries that have different skeletons (or one of the branches must be empty). Each Print statement is associated with a query and only prints values retrieved by its query.

3.2 Notation

SHEAR works with data retrieval programs. To characterize the domain, we reuse three definitions in prior work that targets the same general domain [Shen and Rinard 2019]:

Definition 1. A context $\sigma = \langle \sigma_I, \sigma_D, \sigma_R \rangle \in \mathit{Context}$ (Figure 5) contains value mappings for the input parameters ($\sigma_I \in \mathit{Input}$), database contents ($\sigma_D \in \mathit{Database}$), and results retrieved by database queries ($\sigma_R \in \mathit{Result}$).

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

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.

For the definitions below, we follow the notation in prior work but enhance their expressiveness to work with programs that may contain nested loops and after-loop subprograms.

Definition 4. A *loop layout tree* for a program $P \in$ Prog is a tree that represents information about the execution of loops (Figure 6). Each node in the loop layout 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 ($\text{Tree}_1, \ldots, \text{Tree}_r$) each corresponds to an iteration of the loop. The last subtree (Tree_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 obtained from a query $Q \in$ Query. The first component is the query Q. 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) λ = AfterLoop which indicates execution of the subprogram that follows the loop. If no such loop was found then λ = 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 the corresponding query in W, each row count in t matches the corresponding requirement in W, and each after-loop status in t matches the corresponding flag in W.

3.3 Active Loop Detection

When Shear executes the program, the program interacts with an external database through the Shear proxy. The proxy transparently intercepts the SQL queries, as well as the corresponding retrieved data, and relays them between the program and the database. Figure 3 outlines the system architecture.

Shear detects potential loops in the 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. After a small number of such altered program executions, Shear accurately detects loops at this location in the trace.

We present the Shear loop detection algorithm in Algorithm 4. The DetectLoops procedure first executes the program to collect an *intact* trace. For each query in the trace, it invokes DetectLoopAtQuery (Algorithm 2) to infer if a loop iterates over the query. If a loop is found, DetectLoops invokes DetectLoopIters (Algorithm 3) to identify loop iteration boundaries in the trace. Both of these procedures take advantage of the Shear proxy that alters database traffic during program execution (Algorithm 1). The DetectLoops procedure finally invokes BuildLoopLayoutTree to construct a tree that represents the structure of all loops detected.

3.3.1 Manipulating Database Traffic with Proxy. Shear manipulates the database traffic through its proxy (Figure 3). 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 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.

ExecuteAndPickRows: Algorithm 1 presents the procedure for executing the program, altering its database traffic during execution. The ExecuteAndPickRows procedure takes a program, a context, an integer k, and a list of distinct integer row indices ρ. The procedure executes the program with the provided context while intercepting the database traffic.

For the first $(k-1)$ queries that the program sends to the database, the Shear 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 1). The sub-list of rows are 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 simple 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.

3.3.2 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.

DetectLoopAtQuery: Algorithm 2 infers whether a loop iterates over a specific query during an execution of the program. The DetectLoopAtQuery 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 ExecuteAndPick-Rows for three times. Each invocation executes the program with context σ, during which time it alters the k-th query's database traffic. 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 (line 2 of Algorithm 2). The second execution alters the k-th query to retrieve only the second row (line 3). The third execution alters the k-th query to retrieve only the first two rows (line 4). 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 DetectLoopAtQuery procedure uses these three altered lists to infer whether the program contains a loop that iterates over the specified query — the k-th query in the trace from executing with σ. The procedure obtains the three list suffixes starting from the $(k+1)$-th query (lines 5,6,7). 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

Algorithm 1 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 ExecuteAndPickRows(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
```

(line 9), the number for only the second hypothetical iteration (line 10), and the number for both the first and second iterations (line 11). The procedure then locates the queries that would have been produced by these hypothetical loop iterations (lines 12,13) and by any remaining queries in the program following the hypothetical loop (lines 14,15,16).

Finally, the DetectLoopAtQuery procedure checks if the lengths and contents for these hypothetical iterations are consistent (line 17). 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.

The DetectLoopAtQuery procedure detects a loop if and only if the hypothetical loop is consistent with the three (altered) executions. We show that Algorithm 2 accurately detects the potential loop at the specified query (Section 3.5).

DetectLoopIters: When a loop has been detected, DetectLoops invokes the DetectLoopIters procedure (Algorithm 3) to identify the boundaries for each loop iteration. The DetectLoopIters

Algorithm 2 Detect if a loop iterates over a query by manipulating the database traffic in three program executions

Input: $\boxed{P}$ is the executable of a program $P \in \mathcal{S}$.
Input: σ is a context.
Input: 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** DetectLoopAtQuery($\boxed{P}, \sigma, k$)
2: $s_1 \leftarrow$ ExecuteAndPickRows($\boxed{P}, \sigma, k, [1]$)
3: $s_2 \leftarrow$ ExecuteAndPickRows($\boxed{P}, \sigma, k, [2]$)
4: $s_{12} \leftarrow$ ExecuteAndPickRows($\boxed{P}, \sigma, k, [1, 2]$)
5: $s'_1 \leftarrow s_1[k+1, \ldots]$
6: $s'_2 \leftarrow s_2[k+1, \ldots]$
7: $s'_{12} \leftarrow s_{12}[k+1, \ldots]$
8: $l_\alpha \leftarrow$ Len(s'_1) + Len(s'_2) − Len(s'_{12}) ▷ Length after hypothetical loop
9: $l_1 \leftarrow$ Len(s'_1) − l_α ▷ Length of the first hypothetical iteration
10: $l_2 \leftarrow$ Len(s'_2) − l_α ▷ Length of the second hypothetical iteration
11: $l_{12} \leftarrow$ Len(s'_{12}) − l_α ▷ Length of both hypothetical iterations
12: $\beta_1 \leftarrow s'_1[1, \ldots, l_1]$ ▷ Queries in the first hypothetical iteration
13: $\beta_{12} \leftarrow s'_{12}[1, \ldots, l_{12}]$ ▷ Queries in both hypothetical iterations
14: $\alpha_1 \leftarrow s'_1[l_1 + 1, \ldots]$ ▷ Queries after the hypothetical loop
15: $\alpha_2 \leftarrow s'_2[l_2 + 1, \ldots]$ ▷ Queries after the hypothetical loop
16: $\alpha_{12} \leftarrow s'_{12}[l_{12} + 1, \ldots]$ ▷ Queries after the hypothetical loop
17: **if** $\alpha_1 = \alpha_2 = \alpha_{12}$ **and** PerfectPrefix(β_1, β_{12}) **then**
18: **return** $\langle$`true`, $l_\alpha\rangle$ ▷ Traces consistent with hypothetical loop
19: **end if**
20: **return** $\langle$`false`, Nil$\rangle$ ▷ Traces inconsistent with hypothetical loop
21: **end procedure**

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 3.1), the loop has r iterations each accessing one row from the k-th query. The integer l_α is a result from DetectLoopAtQuery (Algorithm 2) and equals the number of queries in the trace that are produced by the subprogram that follows the detected loop.

The procedure DetectLoopIters identifies the length of each loop iteration in the trace. Specifically, the procedure invokes ExecuteAndPickRows for r times (line 4 of Algorithm 3).[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

[1] A straightforward optimization is to avoid repeatedly executing the program with the same context and alterations.

Algorithm 3 Detect the length of every loop iteration

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

```
1: procedure DetectLoopIters(⌈P⌉, σ, k, r, l_α)
2:     l_β ← Empty list
3:     for i ← 1, ..., r do
4:         s_i ← ExecuteAndPickRows(⌈P⌉, σ, k, [i])
5:         l_i ← Len(s_i) − k − l_α                ▷ Length of the i-th iteration
6:         Append l_i to l_β
7:     end for
8:     return l_β
9: end procedure
```

l_α queries that are produced by the subprogram in P after the loop. The procedure calculates the number of queries in each loop iteration (line 5), then returns the lengths of all iterations as a list l_β (line 8). Using this list of lengths, it is straightforward to divide the intact execution trace (line 3 of Algorithm 4) into segments that correspond to the individual loop iterations.

3.3.3 Detecting and Representing All Looping Structures. The algorithms above work well for programs that may contain multiple loops, nested or otherwise.
DetectLoops: The DetectLoops procedure (Algorithm 4) invokes DetectLoopAtQuery and DetectLoopIters as needed for every query in an execution trace. Programs in the Shear DSL has 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 3.4.2) in any intact trace.
BuildLoopLayoutTree: The BuildLoopLayoutTree 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 the innermost loops.

3.4 Shear Inference Algorithm

Shear infers the control structures of a program executable, $\boxed{P}$, by recursively inferring the AST nodes in a hypothetical program in the Shear DSL. The hypothetical DSL program has the same externally visible behavior as $\boxed{P}$. The recursive algorithm visits each potential path in the

Algorithm 4 Detect loops and build loop layout tree

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

```
1:  procedure DetectLoops(⌈P⌉, σ, e)
2:      L ← Empty list
3:      (Q_1, r_1), ..., (Q_n, r_n) ← e
4:      for k = 1, ..., n do
5:          if r_k ≥ 2 then
6:              ⟨f, l_α⟩ ← DetectLoopAtQuery(⌈P⌉, σ, k)
7:              if f then
8:                  l_β ← DetectLoopIters(⌈P⌉, σ, k, r_k, l_α)
9:                  Append ⟨k, l_β⟩ to L
10:             end if
11:         end if
12:     end for
13:     return BuildLoopLayoutTree(e, L)
14: end procedure
```

hypothetical program AST and expands the Prog nonterminals from top to bottom (Konure [Shen and Rinard 2019] also uses this same top-down inference strategy). During this process, Shear infers looping structures using active loop detection.

After inferring the program functionality as a DSL program, Shear translates the DSL program into a new implementation. Motivations of this regeneration step include program migration, inserting systematic checks, program comprehension, debugging, reverse engineering, and re-engineering of legacy systems.

3.4.1 Recursive Inference Algorithm. We take the existing approach that starts with an initial trivial execution of $\boxed{P}$. The initial execution trace is the starting point with which we generate path constraints to reveal increasingly more interesting behavior of $\boxed{P}$. During this exploration, Shear integrates the looping structures that it infers in each execution trace. The result is successful inference of programs in the Shear DSL that may contain complex looping and repetition structures.
Infer: The entry point to the inference algorithm is the Infer procedure (Algorithm 5). The procedure takes an executable program $\boxed{P}$. It first configures an initial context σ where all database tables are empty and the input parameters are distinct. It then invokes GetTrace to obtain an initial annotated trace t. Finally, Infer invokes InferProg to infer P.
GetTrace: Shear invokes active loop detection in the GetTrace procedure (Algorithm 6). This procedure executes a program, intercepts the database traffic, detects loops in the traffic, de-duplicates the traffic into annotated traces, and identifies one of these annotated traces that satisfy a provided path constraint. The procedure takes an executable program $\boxed{P}$, a path constraint W, and a context σ as parameters. It first runs $\boxed{P}$ in context σ by populating the specified database contents and executing $\boxed{P}$ with the specified inputs. Shear uses its proxy to collect the database traffic during program execution, then converts it into a list e of query-result pairs. The procedure then invokes DetectLoops (Algorithm 4) to obtain the loop layout tree l. Invoking loop detection

requires passing in the executable program $\boxed{P}$ and σ as parameters, as they are used to generate the altered execution traces in DetectLoops. The GetTrace procedure then traverses each path in l from root to a leaf to generate an annotated trace. Among all of the generated traces, the procedure chooses one that is consistent with the path constraint W as the return value.

Algorithm 5 Infer an executable program

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

```
1: procedure Infer(⌈P⌉)
2:     σ ← Database empty, input parameters distinct
3:     t ← GetTrace(⌈P⌉, Nil, σ)
4:     return InferProg(⌈P⌉, Nil, t)
5: end procedure
```

Algorithm 6 Execute a program and deduplicate the trace according to a path constraint

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

```
1: procedure GetTrace(⌈P⌉, W, σ)
2:     e ← Execute(⌈P⌉, σ)
3:     l ← DetectLoops(⌈P⌉, σ, e)
4:     return MatchPath(l, W)
5: end procedure
```

InferProg: Shear infers sophisticated looping and repetition structures in $\boxed{P}$ in the InferProg procedure (Algorithm 7). 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 the next Prog nonterminal to resolve. To resolve the next 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 (so that the loop can be detected) 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. A straightforward optimization is to reuse the traces previously collected, rather than solving for and executing a new context. 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 3.4.2.

If Shear does not detect a loop in t_2 that iterates over the rows retrieved by Q, Shear next 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.

MakePathConstraint: The MakePathConstraint procedure takes a trace prefix s_1, the subsequent 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 any satisfying context must enable the program to execute down the same path as s_1, then perform query Q and retrieve a certain number of rows as specified by i. 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 (Any). The path constraint uses d and a to specify the known looping structures in the trace prefix s_1 followed by Q.

SolveAndGetTrace: The SolveAndGetTrace procedure takes an executable program $\boxed{P}$ and a path constraint W. The procedure uses an SMT solver to obtain a context σ that causes $\boxed{P}$ to produce a trace t that satisfies W if W is satisfiable. The procedure then invokes GetTrace to execute the program, detect loops in the traces, convert the loop layout tree into annotated traces, and obtain a satisfying trace.

3.4.2 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 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 from Section 3.3 that DetectLoops detects loops in the program as long as the loop iterated at least twice (Algorithm 4). When a loop iterated only once, the 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

Algorithm 7 Recursively infer a subprogram

Input: $\boxed{P}$ is the executable of a program $P \in \mathcal{S}$.
Input: s_1 is a prefix of an annotated trace.
Input: s_2 is 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 ▷ Prog := ϵ
  end if
  $k \leftarrow$ 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** ▷ Satisfiable
      $t_{i,1} \leftarrow t_i[1, \ldots, (k+1)]$ ▷ New trace prefix
      $t_{i,2} \leftarrow t_i[(k+2), \ldots]$ ▷ New trace suffix
    end if
  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** (($t_{0,2}$ = 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

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 ExecuteAndPickRows with an empty list ρ. In the altered execution, the query Q is altered to retrieve zero rows. The corresponding loop iterates

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

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

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

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

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

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

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

Fig. 7. Semantics for executing a program using a context to directly obtain a list of query-result pairs

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. 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.

3.5 Soundness Proof Outline

We first outline a soundness proof for the SHEAR loop detection algorithm (Algorithm 4). In the following three definitions, we reuse notation from prior work [Shen and Rinard 2019, 2021], but with enhanced capabilities to express complex looping structures in the SHEAR DSL.

Definition 8. For a program $P \in \text{Prog}$ and a context $\sigma \in \textit{Context}$, $\sigma \vdash P \Downarrow_{\text{exec}} e$ denotes evaluating P in σ to obtain a list of query-result pairs e. $\sigma \vdash P \Downarrow_{\text{loops}} l$ denotes evaluating P in σ to obtain a loop layout tree l. Figure 7 and Figure 8 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 4) 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`

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

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

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

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

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

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

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

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

Fig. 8. Semantics for executing a program using a context to obtain a loop layout tree

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.

We next define key concepts for characterizing active loop detection in Shear.

Definition 11. For program $P \in \text{Prog}$ and context $\sigma \in \textit{Context}$, $\sigma[\mapsto_{:k} P]$ denotes the updated context after evaluating P in σ for the first k queries. Figure 9 defines this concept. $\textsc{Len}_{\text{exec}}(P, \sigma)$ denotes the length of the trace obtained 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 10 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 $P_1, P_2 \in \text{Prog}$: $\sigma \vdash P \Downarrow_{\text{exec}} (Q_1, r_1), \ldots, (Q_n, r_n)$,

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

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

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

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

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

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

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

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

Fig. 9. Updating a context after evaluating a program prefix

$1 \le k \le n$, 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 11 and Figure 12 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 13 defines this concept.

Definition 15. For program $P \in \text{Prog}$, context $\sigma \in \textit{Context}$, integer k where a loop iterates over the k-th query, and list of integers $\rho_1, \ldots, \rho_m$, $\sigma \vdash_{k:} P \Downarrow^{\rho_1,\ldots,\rho_m}_{\text{alter}} e$ 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 14 defines this concept.

As we presented in Section 3.3, a key idea of the SHEAR loop detection algorithm is to manipulate the program's database traffic during executions to observe if the altered behaviors match hypothetical loop structures.

$$\frac{}{P \;-\; \text{Nil} \;=\; P} \tag{nil}$$

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

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

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

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

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

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

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

Fig. 10. Use a list of query-result pairs to consume a program prefix and obtain the remaining subprogram

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 $\textsc{ExecuteAndPickRows}(\boxed{P}, \sigma, k, []) = [q_1, \ldots, q_{n'}]$, 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 $\textsc{ExecuteAndPickRows}(\boxed{P}, \sigma, k, [\rho_1, \ldots, \rho_m]) = [q_1, \ldots, q_{n'}]$, 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 $\textsc{DetectLoopAtQuery}(\boxed{P}, \sigma, k) = \langle \texttt{true}, \textsc{Len}(e') \rangle$ where $\sigma \vdash_{k:} P \Downarrow_{\text{after}} e'$.

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

Theorem 2. For any program $P \in \mathcal{S}$, context $\sigma \in \mathit{Context}$, 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 $\textsc{DetectLoopAtQuery}(\boxed{P}, \sigma, k) = \langle \texttt{false}, \text{Nil} \rangle$.

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 $\textsc{DetectLoopIters}(\boxed{P}, \sigma, k, r, \textsc{Len}(e')) = [\textsc{Len}(e_1), \ldots, \textsc{Len}(e_r)]$.

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

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

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

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

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

Fig. 11. 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. 12. Check if a loop iterates over a specific query and, if so, calculate each iteration's corresponding query-result pairs

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

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

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

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

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

Fig. 13. 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. 14. Semantics for executing a program while altering a query to retrieve only the specified rows

These results show that the active loop detection technique in Section 3.3 is guaranteed to detect loops precisely and return the correct loop layout trees for any program in the Shear DSL. These guarantees hold even when the Shear DSL contains complex looping structures, such as nested and consecutive loops, and allows arbitrary repetitions along execution paths.

The remainder of the soundness proof is about the recursive inference algorithm in Section 3.4. Because this part of the algorithm is adapted from the recursion framework in prior work [Shen and Rinard 2019, 2021], the corresponding proof is analogous to that of prior work. Like prior work, we prove the following theorem for programs whose externally visible behavior is in $\mathcal{S}$ with no Print statements. The main difference here 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.

The Shear 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.

4 EXPERIMENTAL RESULTS

We evaluate the Shear implementation 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. We present more detail and all of the regenerated programs in Appendix A.

- **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 Javacript, 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, Javacript, 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 Javacript, 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 Javacript, 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.

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]. We also developed a synthetic test suite to highlight the capabilities of Shear:

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

Tables 1 and 2 present the experimental results from using the Shear implementation to infer and regenerate the benchmark commands outlined above. Table 1 presents a summary of the programs that Shear regenerated for these commands. The first column (**Command**) presents the name of the command. We highlight commands that are out of the scope of Konure [Shen and Rinard 2019] with the "+" sign. The second column (**Reason**) describes the reasons why a command is out of the scope of Konure and other loop detection techniques:

- **Reason codes "R" and "r"** denote that there are repetitive queries that violate the heuristic assumptions in other systems. "R" denotes that there are non-adjacent repetitions in the trace that violate the assumptions. Commands with this code are out of the scope of Konure and Kobayashi's algorithm [Kobayashi 1984]. "r" denotes that there are adjacent repetitive queries in the trace that violate the assumptions. Commands with this code are out of the scope of Konure, DaViS [Noughi et al. 2014], and Kobayashi's algorithm.
- **Reason code "C"** denotes that a loop body contains structures such as conditional statements or loops. Commands with this code are out of the scope of DaViS and Kobayashi's algorithm.
- **Reason code "N_i"** denotes that the regenerated program has i layers of nested loops. Commands with this code are out of the scope of Konure, DaViS, and Kobayashi's algorithm.
- **Reason code "A"** denotes that there are after-loop subprograms. **Reason code "A_i"** denotes that there are i loops one after the other. Commands with these codes are out of the scope of Konure.

Compared to these other systems, Shear infers and regenerates a substantially more complex set of looping and repetition structures: There are 25 commands (15 from open source applications) out of the scope of Konure. There are 18 commands (14 open source) out of the scope of DaViS. There are 25 commands (15 open source) out of the scope of Kobayashi's algorithm.

The third column (**App**) presents the name of the application to which the command belongs. The fourth column (**In**) presents the number of input parameters for the command. The remaining columns present statistics from the regenerated Python implementations. 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.

We compared the regenerated programs for the commands that are used in both our evaluation and Konure's evaluation. 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.

Compared to Konure, there are 19 commands (13 open source) whose regenerated programs have loops that are supported by Shear but are out of the scope of Konure. Among the commands whose regenerated programs have loops, Shear supports all 25 (19 open source) while Konure supports only 6 (6 open source). There are another 6 commands (2 open source) whose regenerated programs do not have loops that are supported by Shear but are out of the scope of Konure. The capabilities to work with sophisticated looping and repetition structures enables Shear to support commands in real-world applications that are larger and more complex. Notably, four of the RailsCollab commands supported by Shear have between 74 and 97 lines of computation code in the regenerated program. In contrast, the highest corresponding number for Konure is only 38.

Table 2 presents a summary of the inference effort. The first column (**Command**) presents the name of the command. We highlight commands that are out of the scope of Konure with the "+" sign. The second column (**Launch**) presents the average time required to tear down, restart, and execute the application (or its web server) in the Shear environment.

The third column (**Runs**) presents the number of executions that Shear used to infer the command. All commands require less than 440 executions to obtain a model for the command as expressed in the Shear DSL. 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 number of intact executions is positively correlated with the size of the program and the need for disambiguation. The need for disambiguation is positively correlated with the size of the program, the number of outputs, and the number of columns in the database schema. The intact execution numbers are slightly higher than the corresponding numbers reported in Konure (Konure has only intact executions). We attribute this difference to the different implementations for the disambiguation solver in these two systems. 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. Most commands require altered executions less than 7 times the number of intact executions.

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 number of solves is positively correlated with the size of the program, the number of program executions, and the need for disambiguation. These numbers are roughly in the same range as the corresponding numbers reported in Konure. The next column (**Total**) presents the wall-clock time required to infer each program. The times are generally shorter than Konure for commands that required many solves, such as `get_projects_id`, `get_projects_id_users`, `get_channels`, and `get_channels_id_activities`. We attribute this difference to the different implementations for the disambiguation solver in these two systems. The 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 program executions by Shear for detecting more expressive looping and repetition structures. The last column (**Algo Time**) 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 program (which is the product of the launch time and the number of runs). Most of the pure inference time was spent on disambiguation, which is also the case as reported in Konure. The pure inference time is positively correlated with the number of solves and

the size of the program. The pure inference time is between 18–82% of the total wall-clock time for RailsCollab commands, between 7–20% for Kanban and Kandan commands, and between 4–8% for Todo and Enki commands. Overall, the larger a program, the higher percentage of the time was spent on disambiguation and hence the smaller percentage was spent on program executions.

We measured time on a Ubuntu 16.04 virtual machine 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.

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 [2] 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.

5 ADAPTATIONS FOR OTHER CONTEXTS

The current Shear implementation targets computations that iterate over collections of rows returned from database queries. We next discuss how the instrumentation in Shear can be adapted to work with collections that appear in other contexts.

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. JavaScript proxy objects are capable of altering the return values for accesses on object properties during program execution [Jav 2019].

Potential loops in the Mimic paper [Heule et al. 2015] that may benefit from active loop detection include the functions every, filter, forEach, indexOf, map, reduce, some, max, min, and sum. 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.

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]. This implementation collects SQL traces by using Java AspectJ advice [Kiczales et al. 2001] to record information around the execution points where the program performs SQL queries. An AspectJ advice is capable of altering function arguments during program execution [Kiczales et al. 2001].

[2] 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.

DaViS detects potential loops whose loop body has only one SQL query that references the query preceding the loop. 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.

Adaptations for Nero: Nero [Wu 2018] is designed to synthesize database-backed 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. All these accesses are in the form of function calls. The recorded functions include the function `__iter__()`, which returns an iterator for a data structure, and the function `__next__()`, which returns the next element from an iterator. Wrappers of this form are capable of altering the function return values during program execution.

Nero works with loops that iterate over Python lists and dictionaries. In these loops, different iterations are independent from each other. 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.

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]. Emulators of this form are capable of altering the program state during program execution.

Dispatcher uses two loop detection methods, among which the "dynamic" method is based on detecting repetitions in a trace [Kobayashi 1984]. Loops in the application often arise from processing sequences of data fields, 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]. 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.

Active loop detection can benefit all of these systems by enabling them to support a wider range of programs.

6 RELATED WORK

Detecting loops from system traces has been a recurring problem in multiple research areas, including program synthesis, program comprehension, performance profiling and optimization, 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 based on memory access traces, but it does not guarantee identifying the correct loops. Nero [Wu 2018] uses instrumentation to identify where each loop iteration starts in the trace, but it does not identify where the last loop iteration ends unless it already knows the loop body. Among the two loop detection methods in Dispatcher [Caballero et al. 2009; Caballero and Song 2013], the "dynamic" technique detects repetitions from traces using heuristics and it 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

only one SQL query. Kobayashi's loop detection algorithm [Kobayashi 1984] is based on repetitions and it would not work with programs where a loop body contains conditionals. It also would not work with programs that contain ambiguous repetitive instructions.

Konure [Shen and Rinard 2019] 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 (as illustrated by the experimental results in Section 4).

Synthesizing loops from observed executions: Program synthesis is currently an active research area [Alur et al. 2013; Beyene et al. 2015; Bornholt and Torlak 2017; Chasins and Phothilimthana 2017; Ellis et al. 2016; Feng et al. 2018, 2017; Feser et al. 2015; Gulwani et al. 2017; Jeon et al. 2015; Jha et al. 2010; Loncaric et al. 2018; Phothilimthana et al. 2019; Polikarpova et al. 2016; Pu et al. 2018; Si et al. 2018; Solar-Lezama et al. 2006; Wang et al. 2017, 2018, 2019; Yaghmazadeh et al. 2016, 2018, 2017]. We identify a few systems that can synthesize loops by observing executions of an existing program (without accessing its source code) [Biermann et al. 1975; Heule et al. 2015; Qi et al. 2012; Shen and Rinard 2019]. In contrast to these systems, Shear (1) manipulates the database traffic during program execution, as opposed to only running the program from start to end, (2) fully and precisely identifies loop structures in the program, and (3) directly calculates the loop structures based on several executions of the program, instead of searching over multiple candidate solutions.

Synthesizing regular expressions: It is possible to frame some aspects of loop detection using regular expressions. A fundamental difference between Shear and previous regular expression synthesis algorithms is that Shear observes and intervenes in interactions between system components to obtain a top-down, more efficient inference algorithm. There are two kinds of regular expression synthesis algorithms. The first works only with positive and negative examples [Chen et al. 2020; Lee et al. 2016]. These algorithms provide no guarantee 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 and does not work with or require negative examples. The second kind of regular expression synthesis algorithm does deliver a single correct regular expression, but requires 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: There are many techniques that 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 of the programs or execution platforms. Shear observes only the SQL queries, which are largely independent from the low-level implementation details of the program, and detects loops based merely on how these queries change when the algorithm manipulates the data retrieved for certain queries. As a result Shear works well with applications written in any language or any implementation styles, as long as their externally visible behavior conforms to the Shear DSL.

Detecting loops with static analysis or program state: There are many techniques that detect potential loops during program execution according to control flow graphs, instruction addresses,

memory addresses, stack frames, register values, taints, or other forms of low-level runtime information [Caballero et al. 2009; Caballero and Song 2013; Caballero et al. 2007; Carbin et al. 2011; Hayashizaki et al. 2011; Kling et al. 2012; 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 and observes only the SQL queries visible in the network traffic as the program communicates with an external database. As a result SHEAR works well with applications written in any language or any implementation styles, as long as their externally visible behavior conforms to the SHEAR DSL.

7 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 1. Comparison of SHEAR and Konure Loop Detection Algorithms

Command	Reason	App	In	Q	If	For	Out	LoC	Comp
+ **get_projects_id_messages**	R, C, A_2	RailsCollab	2	34	9	3	39	114	75
+ **get_projects_id_messages_id**	r	RailsCollab	3	21	5	0	30	69	39
+ **get_projects_id_messages_display_list**	R, C, A_2	RailsCollab	2	35	9	3	40	115	75
+ **get_projects_id_times** (fixed 500 error)	R, C, A	RailsCollab	2	38	11	1	40	114	74
+ **get_projects_id_times_id**	R	RailsCollab	3	55	15	0	50	147	97
+ **get_projects_id_milestones_id**	R, r, A	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**	C, N_3	Kanban	1	9	1	3	27	55	28
+ **get_api_lists_id**	C, N_2	Kanban	2	9	2	2	27	54	27
+ **get_api_cards**	C, N_2	Kanban	1	8	1	2	23	46	23
get_api_cards_id		Kanban	2	8	2	1	23	45	22
+ **get_api_boards_id**	C, N_3, A	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	C	Todo	0	5	1	1	5	20	15
+ **get_lists_id_tasks**	C, A	Todo	1	6	1	1	2	17	15
+ **get_lists_id_tasks** (fixed 404 error)	C, A	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	C	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**	A	Enki	0	9	1	1	8	26	18
+ **get_archives**	C, A	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)	A_2	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**	r	Synthetic	0	3	0	0	0	7	7
+ **repeat_3**	R, r	Synthetic	0	4	0	0	0	8	8
+ **repeat_4**	R, r	Synthetic	0	5	0	0	0	9	9
+ **repeat_5**	R, r	Synthetic	0	6	0	0	0	10	10
+ **nest**	C, N_2	Synthetic	0	3	0	2	0	15	15
+ **after_2**	R, A_2	Synthetic	0	4	0	2	0	15	15
+ **after_3**	R, A_3	Synthetic	0	6	0	3	0	20	20
+ **after_4**	R, A_4	Synthetic	0	8	0	4	0	25	25
+ **after_5**	R, A_5	Synthetic	0	10	0	5	0	30	30
+ **example** (Section 2)	R, A	Synthetic	1	6	1	1	6	22	16

Table 2. SHEAR Inference Effort

Command	Launch	Runs	= Intact + Alt	Solves	Total	Algo Time
+ **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

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.

Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. 1–8.

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

Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2015. Recursive Games for Compositional Program Synthesis. In *Verified Software: Theories, Tools, and Experiments - 7th International Conference, VSTTE 2015, San Francisco, CA, USA, July 18-19, 2015. Revised Selected Papers*. 19–39.

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 Bornholt and Emina Torlak. 2017. Synthesizing Memory Models from Framework Sketches and Litmus Tests. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. ACM, New York, NY, USA, 467–481. https://doi.org/10.1145/3062341.3062353

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

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

Sarah Chasins and Phitchaya Mangpo Phothilimthana. 2017. Data-Driven Synthesis of Full Probabilistic Programs. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 279–304.

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

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

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. 2016. Sampling for Bayesian Program Learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 1289–1297.

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

Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. 2018. Program synthesis using conflict-driven learning. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA,*

USA, June 18-22, 2018. 420–435.

Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. 2017. Component-based synthesis of table consolidation and transformation tasks from examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*. 422–436.

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.

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. 2017. Program Synthesis. *Foundations and Trends in Programming Languages* 4, 1-2 (2017), 1–119.

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.

Jinseong Jeon, Xiaokang Qiu, Jeffrey S. Foster, and Armando Solar-Lezama. 2015. JSketch: sketching for Java. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. 934–937.

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

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

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

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

Barbara Liskov, Alan Snyder, Russell Atkinson, and Craig Schaffert. 1977. Abstraction Mechanisms in CLU. *Commun. ACM* 20, 8 (Aug. 1977), 564–576. https://doi.org/10.1145/359763.359789

Calvin Loncaric, Michael D. Ernst, and Emina Torlak. 2018. Generalized Data Structure Synthesis. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. ACM, New York, NY, USA, 958–968. https://doi.org/10.1145/3180155.3180211

Fan Long, Stelios Sidiroglou-Douskos, and Martin Rinard. 2014. Automatic Runtime Error Repair and Containment via Recovery Shepherding. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Edinburgh, United Kingdom) *(PLDI '14')*. ACM, New York, NY, USA, 227–238. https://doi.org/10.1145/2594291.2594337

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

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

Phitchaya Mangpo Phothilimthana, Archibald Samuel Elliott, An Wang, Abhinav Jangda, Bastian Hagedorn, Henrik Barthels, Samuel J. Kaufman, Vinod Grover, Emina Torlak, and Rastislav Bodik. 2019. Swizzle Inventor: Data Movement Synthesis for GPU Kernels. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 65–78. https://doi.org/10.1145/3297858.3304059

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.

Yewen Pu, Zachery Miranda, Armando Solar-Lezama, and Leslie Kaelbling. 2018. Selecting Representative Examples for Program Synthesis. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*. PMLR, 4161–4170. http://proceedings.mlr.press/v80/pu18b.html

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

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

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

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)*. ACM, New York, NY, USA, 515–527. https://doi.org/10.1145/3236024.3236034

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

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

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. 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. 2018. Program synthesis using abstraction refinement. *PACMPL* 2, POPL (2018), 63:1–63:30.

Yuepeng Wang, James Dong, Rushi Shah, and Isil Dillig. 2019. Synthesizing Database Programs for Schema Refactoring. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. ACM, New York, NY, USA, 286–300. https://doi.org/10.1145/3314221.3314588

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.

Navid Yaghmazadeh, Christian Klinger, Isil Dillig, and Swarat Chaudhuri. 2016. Synthesizing transformations on hierarchically structured data. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*. 508–521.

Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. 2018. Automated Migration of Hierarchical Data to Relational Tables Using Programming-by-example. *Proc. VLDB Endow.* 11, 5 (Jan. 2018), 580–593. https://doi.org/10.1145/3187009.3177735

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: Query Synthesis from Natural Language. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 63 (Oct. 2017), 26 pages. https://doi.org/10.1145/3133887

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.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX, San Jose, CA, 15–28. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia

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
                        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(s18, '
                                messages', '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(s18, 'messages', 'project_id')})
                    s18 = s18_all
                    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]})
                    outputs.extend(util.get_data(s25, 'people', 'project_id'))
                    outputs.extend(util.get_data(s25, 'people', 'user_id'))
                    s26 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`
                        WHERE `messages`.`project_id` = :x0 AND `messages`.`
                        is_important` = 1", {'x0': inputs[1]})
                    s27 = util.do_sql(conn, "SELECT COUNT(*) FROM `categories`
                        WHERE `categories`.`project_id` = :x0", {'x0': inputs
                        [1]})
                    if util.has_rows(s27):
                        s28 = util.do_sql(conn, "SELECT `categories`.* FROM `
                            categories`  WHERE `categories`.`project_id` = :x0",
                             {'x0': inputs[1]})
                        outputs.extend(util.get_data(s28, 'categories', 'id'))
                        outputs.extend(util.get_data(s28, 'categories', '
                            project_id'))
                        outputs.extend(util.get_data(s28, 'categories', 'name'))
                        s28_all = s28
```

```
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]})
83                      if util.has_rows(s11):
84                          s12 = util.do_sql(conn, "SELECT `categories`.* FROM `
                                categories`  WHERE `categories`.`project_id` = :x0",
                                 {'x0': inputs[1]})
85                          outputs.extend(util.get_data(s12, 'categories', 'id'))
86                          outputs.extend(util.get_data(s12, 'categories', '
                                project_id'))
87                          outputs.extend(util.get_data(s12, 'categories', 'name'))
88                          s12_all = s12
89                          for s12 in s12_all:
90                              s13 = util.do_sql(conn, "SELECT  `projects`.* FROM `
                                    projects`  WHERE `projects`.`id` = :x0 LIMIT 1",
                                     {'x0': util.get_one_data(s12, 'categories', '
                                    project_id')})
91                              outputs.extend(util.get_data(s13, 'projects', 'id'))
92                              outputs.extend(util.get_data(s13, 'projects', 'name'
                                    ))
93                              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')})
94                              outputs.extend(util.get_data(s14, 'people', '
                                    project_id'))
95                              outputs.extend(util.get_data(s14, 'people', 'user_id
                                    '))
96                              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')})
97                              outputs.extend(util.get_data(s15, 'people', '
                                    project_id'))
98                              outputs.extend(util.get_data(s15, 'people', 'user_id
                                    '))
99                              if util.has_rows(s15):
100                                 pass
101                             else:
102                                 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'
                                        )})
103                                 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'
                                        )})
104                         s12 = s12_all
105                         pass
106                     else:
107                         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):
            s5 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`
                WHERE `companies`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(
                s0, 'users', 'company_id')})
            if util.has_rows(s5):
                s6 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`
                    WHERE `companies`.`id` IS NULL LIMIT 1", {})
                s7 = util.do_sql(conn, "SELECT  `messages`.* FROM `messages`
                    WHERE `messages`.`project_id` = :x0 AND `messages`.`id` = :
                    x1 ORDER BY created_on DESC LIMIT 1", {'x0': inputs[1], 'x1'
                    : inputs[2]})
                outputs.extend(util.get_data(s7, 'messages', 'id'))
                outputs.extend(util.get_data(s7, 'messages', 'project_id'))
                outputs.extend(util.get_data(s7, 'messages', 'title'))
                outputs.extend(util.get_data(s7, 'messages', 'text'))
                if util.has_rows(s7):
                    s8 = util.do_sql(conn, "SELECT  `projects`.* FROM `projects`
                          WHERE `projects`.`id` = :x0 LIMIT 1", {'x0': inputs
                        [1]})
                    outputs.extend(util.get_data(s8, 'projects', 'id'))
                    outputs.extend(util.get_data(s8, 'projects', 'name'))
                    s9 = util.do_sql(conn, "SELECT  `people`.* FROM `people`
                        WHERE (user_id = :x0 AND project_id = :x1) LIMIT 1", {'
                        x0': util.get_one_data(s0, 'users', 'id'), 'x1': inputs
                        [1]})
                    outputs.extend(util.get_data(s9, 'people', 'project_id'))
```

```
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'))
```

```
                        outputs.extend(util.get_data(s18, 'users', 'display_name
                            '))
                        s19 = util.do_sql(conn, "SELECT  `people`.* FROM `people
                            `  WHERE (user_id = :x0 AND project_id = :x1) LIMIT
                            1", {'x0': util.get_one_data(s0, 'users', 'id'), 'x1
                            ': inputs[1]})
                        outputs.extend(util.get_data(s19, 'people', 'project_id'
                            ))
                        outputs.extend(util.get_data(s19, 'people', 'user_id'))
                        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):
```

```
13              s5 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`
                    WHERE `companies`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(
                    s0, 'users', 'company_id')})
14              outputs.extend(util.get_data(s5, 'companies', 'id'))
15              outputs.extend(util.get_data(s5, 'companies', 'name'))
16              outputs.extend(util.get_data(s5, 'companies', 'homepage'))
17              if util.has_rows(s5):
18                  s6 = util.do_sql(conn, "SELECT  `companies`.* FROM `companies`
                        WHERE `companies`.`id` IS NULL LIMIT 1", {})
19                  s7 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                        messages`.`project_id` = :x0", {'x0': inputs[1]})
20                  s8 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`  WHERE `
                        messages`.`project_id` = :x0", {'x0': inputs[1]})
21                  if util.has_rows(s8):
22                      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]})
23                      outputs.extend(util.get_data(s18, 'messages', 'id'))
24                      outputs.extend(util.get_data(s18, 'messages', 'project_id'))
25                      outputs.extend(util.get_data(s18, 'messages', 'title'))
26                      s18_all = s18
27                      for s18 in s18_all:
28                          s19 = util.do_sql(conn, "SELECT  `projects`.* FROM `
                                projects`  WHERE `projects`.`id` = :x0 LIMIT 1", {'
                                x0': util.get_one_data(s18, 'messages', 'project_id'
                                )})
29                          outputs.extend(util.get_data(s19, 'projects', 'id'))
30                          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:
```

```
                    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')})
                    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')})
                    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')})
                    outputs.extend(util.get_data(s24, 'users', 'id'))
                    outputs.extend(util.get_data(s24, 'users', '
                        display_name'))
                s18 = s18_all
                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]})
                outputs.extend(util.get_data(s26, 'people', 'project_id'))
                outputs.extend(util.get_data(s26, 'people', 'user_id'))
                s27 = util.do_sql(conn, "SELECT COUNT(*) FROM `messages`
                    WHERE `messages`.`project_id` = :x0 AND `messages`.`
                    is_important` = 1", {'x0': inputs[1]})
                s28 = util.do_sql(conn, "SELECT COUNT(*) FROM `categories`
                    WHERE `categories`.`project_id` = :x0", {'x0': inputs
                    [1]})
                if util.has_rows(s28):
                    s29 = util.do_sql(conn, "SELECT `categories`.* FROM `
                        categories`  WHERE `categories`.`project_id` = :x0",
                         {'x0': inputs[1]})
                    outputs.extend(util.get_data(s29, 'categories', 'id'))
                    outputs.extend(util.get_data(s29, 'categories', '
                        project_id'))
                    outputs.extend(util.get_data(s29, 'categories', 'name'))
                    s29_all = s29
                    for s29 in s29_all:
                        s30 = util.do_sql(conn, "SELECT  `projects`.* FROM `
                            projects`  WHERE `projects`.`id` = :x0 LIMIT 1",
                             {'x0': util.get_one_data(s29, 'categories', '
                            project_id')})
                        outputs.extend(util.get_data(s30, 'projects', 'id'))
                        outputs.extend(util.get_data(s30, 'projects', 'name'
                            ))
                        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')})
                        outputs.extend(util.get_data(s31, 'people', '
                            project_id'))
                        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'))
```

```
                            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.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):
```

```
33                          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')})
34                          outputs.extend(util.get_data(s31, 'task_lists',
                                'project_id'))
35                          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')})
36                          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')})
70                                      outputs.extend(util.get_data(s19, 'task_lists',
                                            'project_id'))
```

```
71                          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')})
72                          outputs.extend(util.get_data(s20, 'projects', '
                                id'))
73                          outputs.extend(util.get_data(s20, 'projects', '
                                name'))
74                          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')})
75                          outputs.extend(util.get_data(s21, 'people', '
                                project_id'))
76                          outputs.extend(util.get_data(s21, 'people', '
                                user_id'))
77                          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')})
78                          outputs.extend(util.get_data(s22, 'people', '
                                project_id'))
79                          outputs.extend(util.get_data(s22, 'people', '
                                user_id'))
80                          if util.has_rows(s22):
81                              pass
82                          else:
83                              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')})
84                              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')})
85                      else:
86                          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')})
87                          outputs.extend(util.get_data(s14, 'projects', '
                                id'))
88                          outputs.extend(util.get_data(s14, 'projects', '
                                name'))
```

```
89                                  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')})
90                                  outputs.extend(util.get_data(s15, 'people', '
                                        project_id'))
91                                  outputs.extend(util.get_data(s15, 'people', '
                                        user_id'))
92                                  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')})
93                                  outputs.extend(util.get_data(s16, 'people', '
                                        project_id'))
94                                  outputs.extend(util.get_data(s16, 'people', '
                                        user_id'))
95                                  if util.has_rows(s16):
96                                      pass
97                                  else:
98                                      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')})
99                                      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')})
100                             s10 = s10_all
101                             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]})
102                             outputs.extend(util.get_data(s37, 'people', 'project_id'))
103                             outputs.extend(util.get_data(s37, 'people', 'user_id'))
104                         else:
105                             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]})
106                             outputs.extend(util.get_data(s9, 'people', 'project_id'))
107                             outputs.extend(util.get_data(s9, 'people', 'user_id'))
108                     else:
109                         pass
110                 else:
111                     pass
112             else:
113                 pass
114             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):
```

```
31                              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
                                    ')})
32                              outputs.extend(util.get_data(s47, 'tasks', 'id'))
33                              outputs.extend(util.get_data(s47, 'tasks', 'text'))
34                              if util.has_rows(s47):
35                                  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')})
36                                  outputs.extend(util.get_data(s51, 'task_lists',
                                        'project_id'))
37                                  if util.has_rows(s51):
38                                      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]})
39                                      outputs.extend(util.get_data(s52, 'people',
                                            'project_id'))
40                                      outputs.extend(util.get_data(s52, 'people',
                                            'user_id'))
41                                      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]})
42                                      outputs.extend(util.get_data(s53, 'people',
                                            'project_id'))
43                                      outputs.extend(util.get_data(s53, 'people',
                                            'user_id'))
44                                      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]})
45                                      outputs.extend(util.get_data(s54, 'people',
                                            'project_id'))
46                                      outputs.extend(util.get_data(s54, 'people',
                                            'user_id'))
47                                  else:
48                                      pass
49                              else:
50                                  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]})
51                                  outputs.extend(util.get_data(s48, 'people', '
                                        project_id'))
52                                  outputs.extend(util.get_data(s48, 'people', '
                                        user_id'))
```

```
53                          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]})
54                          outputs.extend(util.get_data(s49, 'people', '
                                project_id'))
55                          outputs.extend(util.get_data(s49, 'people', '
                                user_id'))
56                          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]})
57                          outputs.extend(util.get_data(s50, 'people', '
                                project_id'))
58                          outputs.extend(util.get_data(s50, 'people', '
                                user_id'))
59                  else:
60                      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')})
61                      outputs.extend(util.get_data(s38, 'users', '
                            display_name'))
62                      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
                            ')})
63                      outputs.extend(util.get_data(s39, 'tasks', 'id'))
64                      outputs.extend(util.get_data(s39, 'tasks', 'text'))
65                      if util.has_rows(s39):
66                          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')})
67                          outputs.extend(util.get_data(s43, 'task_lists',
                                'project_id'))
68                          if util.has_rows(s43):
69                              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):
```

```
                    s31 = util.do_sql(conn, "SELECT  `task_lists`.*
                        FROM `task_lists`  WHERE `task_lists`.`id` =
                         :x0 LIMIT 1", {'x0': util.get_one_data(s25,
                         'tasks', 'task_list_id')})
                    outputs.extend(util.get_data(s31, 'task_lists',
                        'project_id'))
                    if util.has_rows(s31):
                        s32 = util.do_sql(conn, "SELECT  `people`.*
                            FROM `people`  WHERE (user_id = :x0 AND
                            project_id = :x1) LIMIT 1", {'x0': util.
                            get_one_data(s0, 'users', 'id'), 'x1':
                            inputs[1]})
                        s33 = util.do_sql(conn, "SELECT `people`.*
                            FROM `people`  WHERE `people`.`user_id`
                            = :x0 AND `people`.`project_id` = :x1",
                            {'x0': util.get_one_data(s0, 'users', '
                            id'), 'x1': inputs[1]})
                        s34 = util.do_sql(conn, "SELECT  `people`.*
                            FROM `people`  WHERE (user_id = :x0 AND
                            project_id = :x1) LIMIT 1", {'x0': util.
                            get_one_data(s0, 'users', 'id'), 'x1':
                            inputs[1]})
                        s35 = util.do_sql(conn, "SELECT `people`.*
                            FROM `people`  WHERE `people`.`user_id`
                            = :x0 AND `people`.`project_id` = :x1",
                            {'x0': util.get_one_data(s0, 'users', '
                            id'), 'x1': inputs[1]})
                        s36 = util.do_sql(conn, "SELECT  `people`.*
                            FROM `people`  WHERE (user_id = :x0 AND
                            project_id = :x1) LIMIT 1", {'x0': util.
                            get_one_data(s0, 'users', 'id'), 'x1':
                            inputs[1]})
                    else:
                        pass
                else:
                    s26 = util.do_sql(conn, "SELECT  `people`.* FROM
                         `people`  WHERE (user_id = :x0 AND
                        project_id = :x1) LIMIT 1", {'x0': util.
                        get_one_data(s0, 'users', 'id'), 'x1':
                        inputs[1]})
                    s27 = util.do_sql(conn, "SELECT `people`.* FROM
                        `people`  WHERE `people`.`user_id` = :x0 AND
                         `people`.`project_id` = :x1", {'x0': util.
                        get_one_data(s0, 'users', 'id'), 'x1':
                        inputs[1]})
                    s28 = util.do_sql(conn, "SELECT  `people`.* FROM
                         `people`  WHERE (user_id = :x0 AND
                        project_id = :x1) LIMIT 1", {'x0': util.
                        get_one_data(s0, 'users', 'id'), 'x1':
                        inputs[1]})
                    s29 = util.do_sql(conn, "SELECT `people`.* FROM
                        `people`  WHERE `people`.`user_id` = :x0 AND
                         `people`.`project_id` = :x1", {'x0': util.
                        get_one_data(s0, 'users', 'id'), 'x1':
                        inputs[1]})
```

```
                                s30 = util.do_sql(conn, "SELECT  `people`.* FROM
                                     `people`  WHERE (user_id = :x0 AND
                                    project_id = :x1) LIMIT 1", {'x0': util.
                                    get_one_data(s0, 'users', 'id'), 'x1':
                                    inputs[1]})
                        else:
                            s12 = util.do_sql(conn, "SELECT  `users`.* FROM `
                                users`  WHERE `users`.`id` = :x0 LIMIT 1", {'x0'
                                : util.get_one_data(s7, 'time_records', '
                                assigned_to_user_id')})
                            outputs.extend(util.get_data(s12, 'users', '
                                display_name'))
                            s13 = util.do_sql(conn, "SELECT  `tasks`.* FROM `
                                tasks`  WHERE `tasks`.`id` = :x0 LIMIT 1", {'x0'
                                : util.get_one_data(s7, 'time_records', 'task_id
                                ')})
                            outputs.extend(util.get_data(s13, 'tasks', 'id'))
                            outputs.extend(util.get_data(s13, 'tasks', 'text'))
                            if util.has_rows(s13):
                                s19 = util.do_sql(conn, "SELECT  `task_lists`.*
                                    FROM `task_lists`  WHERE `task_lists`.`id` =
                                     :x0 LIMIT 1", {'x0': util.get_one_data(s13,
                                     'tasks', 'task_list_id')})
                                outputs.extend(util.get_data(s19, 'task_lists',
                                    'project_id'))
                                if util.has_rows(s19):
                                    s20 = util.do_sql(conn, "SELECT  `people`.*
                                        FROM `people`  WHERE (user_id = :x0 AND
                                        project_id = :x1) LIMIT 1", {'x0': util.
                                        get_one_data(s0, 'users', 'id'), 'x1':
                                        inputs[1]})
                                    s21 = util.do_sql(conn, "SELECT `people`.*
                                        FROM `people`  WHERE `people`.`user_id`
                                        = :x0 AND `people`.`project_id` = :x1",
                                        {'x0': util.get_one_data(s0, 'users', '
                                        id'), 'x1': inputs[1]})
                                    s22 = util.do_sql(conn, "SELECT  `people`.*
                                        FROM `people`  WHERE (user_id = :x0 AND
                                        project_id = :x1) LIMIT 1", {'x0': util.
                                        get_one_data(s0, 'users', 'id'), 'x1':
                                        inputs[1]})
                                    s23 = util.do_sql(conn, "SELECT `people`.*
                                        FROM `people`  WHERE `people`.`user_id`
                                        = :x0 AND `people`.`project_id` = :x1",
                                        {'x0': util.get_one_data(s0, 'users', '
                                        id'), 'x1': inputs[1]})
                                    s24 = util.do_sql(conn, "SELECT  `people`.*
                                        FROM `people`  WHERE (user_id = :x0 AND
                                        project_id = :x1) LIMIT 1", {'x0': util.
                                        get_one_data(s0, 'users', 'id'), 'x1':
                                        inputs[1]})
                                else:
                                    pass
                            else:
```

```
                                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]})
```

```
10          outputs.extend(util.get_data(s4, 'projects', 'id'))
11          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  `milestones`.* FROM `milestones`
                          WHERE `milestones`.`project_id` = :x0 AND `milestones`.`id
                        ` = :x1 LIMIT 1", {'x0': inputs[1], 'x1': inputs[2]})
16                  outputs.extend(util.get_data(s7, 'milestones', 'id'))
17                  outputs.extend(util.get_data(s7, 'milestones', 'project_id'))
18                  outputs.extend(util.get_data(s7, 'milestones', 'name'))
19                  outputs.extend(util.get_data(s7, 'milestones', 'description'))
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                      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]})
28                      outputs.extend(util.get_data(s10, 'people', 'project_id'))
29                      outputs.extend(util.get_data(s10, 'people', 'user_id'))
30                      if util.has_rows(s10):
31                          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')})
32                          outputs.extend(util.get_data(s11, 'companies', 'name'))
33                          if util.has_rows(s11):
34                              s19 = util.do_sql(conn, "SELECT `messages`.* FROM `
                                    messages`  WHERE `messages`.`milestone_id` = :x0
                                    ", {'x0': inputs[2]})
35                              outputs.extend(util.get_data(s19, 'messages', 'id'))
36                              outputs.extend(util.get_data(s19, 'messages', '
                                    milestone_id'))
37                              outputs.extend(util.get_data(s19, 'messages', 'title
                                    '))
38                              s19_all = s19
39                              for s19 in s19_all:
40                                  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')})
41                                  outputs.extend(util.get_data(s20, 'users', 'id')
                                        )
```

```
42                          outputs.extend(util.get_data(s20, 'users', '
                                display_name'))
43                      s19 = s19_all
44                      s21 = util.do_sql(conn, "SELECT `task_lists`.* FROM
                            `task_lists`  WHERE `task_lists`.`milestone_id`
                            = :x0 ORDER BY `order` DESC", {'x0': inputs[2]})
45                      outputs.extend(util.get_data(s21, 'task_lists', 'id'
                            ))
46                      outputs.extend(util.get_data(s21, 'task_lists', '
                            milestone_id'))
47                      outputs.extend(util.get_data(s21, 'task_lists', '
                            name'))
48                      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]})
49                      outputs.extend(util.get_data(s22, 'people', '
                            project_id'))
50                      outputs.extend(util.get_data(s22, 'people', 'user_id
                            '))
51                      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]})
52                      outputs.extend(util.get_data(s23, 'people', '
                            project_id'))
53                      outputs.extend(util.get_data(s23, 'people', 'user_id
                            '))
54                      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]})
55                      outputs.extend(util.get_data(s24, 'people', '
                            project_id'))
56                      outputs.extend(util.get_data(s24, 'people', 'user_id
                            '))
57                  else:
58                      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')})
59                      outputs.extend(util.get_data(s12, 'users', '
                            display_name'))
60                      s13 = util.do_sql(conn, "SELECT `messages`.* FROM `
                            messages`  WHERE `messages`.`milestone_id` = :x0
                            ", {'x0': inputs[2]})
61                      outputs.extend(util.get_data(s13, 'messages', 'id'))
62                      outputs.extend(util.get_data(s13, 'messages', '
                            milestone_id'))
63                      outputs.extend(util.get_data(s13, 'messages', 'title
                            '))
64                      s13_all = s13
65                      for s13 in s13_all:
66                          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):
```

```
6           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')})
7           outputs.extend(util.get_data(s1, 'cards', 'id'))
8           outputs.extend(util.get_data(s1, 'cards', 'list_id'))
9           outputs.extend(util.get_data(s1, 'cards', 'title'))
10          outputs.extend(util.get_data(s1, 'cards', 'description'))
11          outputs.extend(util.get_data(s1, 'cards', 'due_date'))
12          outputs.extend(util.get_data(s1, 'cards', 'position'))
13          s1_all = s1
14          for s1 in s1_all:
15              s2 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `
                    card_comments`.`card_id` = :x0", {'x0': util.get_one_data(s1, '
                    cards', 'id')})
16              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')})
17              outputs.extend(util.get_data(s3, 'lists', 'id'))
18              s4 = util.do_sql(conn, "SELECT  `users`.* FROM `users` WHERE `users
                    `.`id` = :x0  ORDER BY `users`.`id` ASC LIMIT 1", {'x0': util.
                    get_one_data(s1, 'cards', 'assignee_id')})
19              outputs.extend(util.get_data(s4, 'users', 'id'))
20              outputs.extend(util.get_data(s4, 'users', 'email'))
21              outputs.extend(util.get_data(s4, 'users', 'bio'))
22              outputs.extend(util.get_data(s4, 'users', 'full_name'))
23              s5 = util.do_sql(conn, "SELECT `card_comments`.* FROM `card_comments
                    ` WHERE `card_comments`.`card_id` = :x0  ORDER BY `card_comments
                    `.`created_at` DESC", {'x0': util.get_one_data(s1, 'cards', 'id'
                    )})
24              outputs.extend(util.get_data(s5, 'card_comments', 'card_id'))
25              outputs.extend(util.get_data(s5, 'card_comments', 'content'))
26              s5_all = s5
27              for s5 in s5_all:
28                  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')})
29                  outputs.extend(util.get_data(s6, 'cards', 'id'))
30                  outputs.extend(util.get_data(s6, 'cards', 'list_id'))
31                  outputs.extend(util.get_data(s6, 'cards', 'title'))
32                  outputs.extend(util.get_data(s6, 'cards', 'description'))
33                  outputs.extend(util.get_data(s6, 'cards', 'due_date'))
34                  outputs.extend(util.get_data(s6, 'cards', 'position'))
35                  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')})
36                  outputs.extend(util.get_data(s7, 'users', 'id'))
37                  outputs.extend(util.get_data(s7, 'users', 'email'))
38                  outputs.extend(util.get_data(s7, 'users', 'bio'))
39                  outputs.extend(util.get_data(s7, 'users', 'full_name'))
40              s5 = s5_all
41              pass
42          s1 = s1_all
43          pass
44      else:
```

```
45          pass
46      return util.add_warnings(outputs)
```

A.13 Kanban Task Manager Command get_api_cards_id

```
1   def get_api_cards_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  `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]})
7           outputs.extend(util.get_data(s1, 'cards', 'id'))
8           outputs.extend(util.get_data(s1, 'cards', 'list_id'))
9           outputs.extend(util.get_data(s1, 'cards', 'title'))
10          outputs.extend(util.get_data(s1, 'cards', 'description'))
11          outputs.extend(util.get_data(s1, 'cards', 'due_date'))
12          outputs.extend(util.get_data(s1, 'cards', 'position'))
13          if util.has_rows(s1):
14              s2 = util.do_sql(conn, "SELECT COUNT(*) FROM `card_comments` WHERE `
                    card_comments`.`card_id` = :x0", {'x0': inputs[1]})
15              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')})
16              outputs.extend(util.get_data(s3, 'lists', 'id'))
17              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')})
18              outputs.extend(util.get_data(s4, 'users', 'id'))
19              outputs.extend(util.get_data(s4, 'users', 'email'))
20              outputs.extend(util.get_data(s4, 'users', 'bio'))
21              outputs.extend(util.get_data(s4, 'users', 'full_name'))
22              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]})
23              outputs.extend(util.get_data(s5, 'card_comments', 'card_id'))
24              outputs.extend(util.get_data(s5, 'card_comments', 'content'))
25              s5_all = s5
26              for s5 in s5_all:
27                  s6 = util.do_sql(conn, "SELECT  `cards`.* FROM `cards` WHERE `
                        cards`.`id` = :x0  ORDER BY `cards`.`position` ASC LIMIT 1",
                         {'x0': util.get_one_data(s5, 'card_comments', 'card_id')})
28                  outputs.extend(util.get_data(s6, 'cards', 'id'))
29                  outputs.extend(util.get_data(s6, 'cards', 'list_id'))
30                  outputs.extend(util.get_data(s6, 'cards', 'title'))
31                  outputs.extend(util.get_data(s6, 'cards', 'description'))
32                  outputs.extend(util.get_data(s6, 'cards', 'due_date'))
33                  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):
        s1 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`
            id` = :x0  ORDER BY `users`.`id` ASC LIMIT 1", {'x0': util.
            get_one_data(s0, 'users', 'id')})
        outputs.extend(util.get_data(s1, 'users', 'email'))
        s2 = util.do_sql(conn, "SELECT DISTINCT `projects`.* 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(s2, 'projects', 'id'))
        outputs.extend(util.get_data(s2, 'projects', 'name'))
        outputs.extend(util.get_data(s2, 'projects', 'start_date'))
        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')})
        outputs.extend(util.get_data(s3, 'users', 'email'))
        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')})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
        outputs.extend(util.get_data(s4, 'projects', 'name'))
        outputs.extend(util.get_data(s4, 'projects', 'start_date'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.21 Fulcrum Task Manager Command get_projects

```
def get_projects (conn, inputs):
    util.clear_warnings()
    outputs = []
    s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`email
        ` = :x0  ORDER BY `users`.`id` ASC LIMIT 1", {'x0': inputs[0]})
    outputs.extend(util.get_data(s0, 'users', 'email'))
    if util.has_rows(s0):
        s1 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`
            id` = :x0  ORDER BY `users`.`id` ASC LIMIT 1", {'x0': util.
            get_one_data(s0, 'users', 'id')})
```

```
        outputs.extend(util.get_data(s1, 'users', 'email'))
        s2 = util.do_sql(conn, "SELECT DISTINCT `projects`.* 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(s2, 'projects', 'id'))
        outputs.extend(util.get_data(s2, 'projects', 'name'))
        outputs.extend(util.get_data(s2, 'projects', 'start_date'))
        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')})
        outputs.extend(util.get_data(s3, 'users', 'email'))
        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')})
        outputs.extend(util.get_data(s4, 'projects', 'id'))
        outputs.extend(util.get_data(s4, 'projects', 'name'))
        outputs.extend(util.get_data(s4, 'projects', 'start_date'))
    else:
        pass
    return util.add_warnings(outputs)
```

A.22 Fulcrum Task Manager Command get_projects_id

```
def get_projects_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]})
        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')})
                outputs.extend(util.get_data(s7, 'notes', 'id'))
                outputs.extend(util.get_data(s7, 'notes', 'note'))
                outputs.extend(util.get_data(s7, 'notes', 'user_id'))
```

```
                outputs.extend(util.get_data(s7, 'notes', 'story_id'))
            else:
                pass
        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.24 Fulcrum Task Manager Command get_projects_id_stories_id

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

```
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

```
1   def get_projects_id_stories_id_notes (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              if util.has_rows(s6):
13                  s8 = util.do_sql(conn, "SELECT `notes`.* FROM `notes`  WHERE `
                        notes`.`story_id` = :x0", {'x0': inputs[2]})
14                  outputs.extend(util.get_data(s8, 'notes', 'id'))
15                  outputs.extend(util.get_data(s8, 'notes', 'note'))
16                  outputs.extend(util.get_data(s8, 'notes', 'user_id'))
17                  outputs.extend(util.get_data(s8, 'notes', 'story_id'))
18              else:
19                  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')})
20          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
```

```
25      return util.add_warnings(outputs)
```

A.28 Kandan Chat Room Command get_channels

```
1  def get_channels (conn, inputs):
2      util.clear_warnings()
3      outputs = []
4      s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`
           username` = :x0 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 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
7          s2 = util.do_sql(conn, "SELECT `channels`.* FROM `channels`", {})
8          if util.has_rows(s2):
9              s5 = util.do_sql(conn, "SELECT `activities`.* FROM `activities`
                   WHERE `activities`.`channel_id` IN :x0", {'x0': util.get_data(s2
                   , 'channels', 'id')})
10             if util.has_rows(s5):
11                 s10 = util.do_sql(conn, "SELECT `users`.* FROM `users`  WHERE `
                       users`.`id` IN :x0", {'x0': util.get_data(s5, 'activities',
                       'user_id')})
12                 s11 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                       users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, '
                       users', 'id')})
13                 s12 = util.do_sql(conn, "SELECT `channels`.* FROM `channels`",
                       {})
14                 outputs.extend(util.get_data(s12, 'channels', 'id'))
15                 outputs.extend(util.get_data(s12, 'channels', 'name'))
16                 outputs.extend(util.get_data(s12, 'channels', 'user_id'))
17                 s12_all = s12
18                 for s12 in s12_all:
19                     s13 = util.do_sql(conn, "SELECT COUNT(*) FROM `activities`
                           WHERE `activities`.`channel_id` = :x0", {'x0': util.
                           get_one_data(s12, 'channels', 'id')})
20                     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')})
21                     outputs.extend(util.get_data(s14, 'activities', 'id'))
22                     outputs.extend(util.get_data(s14, 'activities', 'content'))
23                     outputs.extend(util.get_data(s14, 'activities', 'channel_id'
                           ))
24                     outputs.extend(util.get_data(s14, 'activities', 'user_id'))
25                     if util.has_rows(s14):
26                         s15 = util.do_sql(conn, "SELECT `users`.* FROM `users`
                               WHERE `users`.`id` IN :x0", {'x0': util.get_data(s14
                               , 'activities', 'user_id')})
27                         outputs.extend(util.get_data(s15, 'users', 'id'))
28                         outputs.extend(util.get_data(s15, 'users', 'email'))
29                         outputs.extend(util.get_data(s15, 'users', 'first_name')
                               )
30                         outputs.extend(util.get_data(s15, 'users', 'last_name'))
31                         outputs.extend(util.get_data(s15, 'users', 'username'))
32                     else:
33                         pass
34                 s12 = s12_all
```

```
35                pass
36            else:
37                s6 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                      users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, '
                      users', 'id')})
38                s7 = util.do_sql(conn, "SELECT `channels`.* FROM `channels`",
                      {})
39                outputs.extend(util.get_data(s7, 'channels', 'id'))
40                outputs.extend(util.get_data(s7, 'channels', 'name'))
41                outputs.extend(util.get_data(s7, 'channels', 'user_id'))
42                s7_all = s7
43                for s7 in s7_all:
44                    s8 = util.do_sql(conn, "SELECT COUNT(*) FROM `activities`
                          WHERE `activities`.`channel_id` = :x0", {'x0': util.
                          get_one_data(s7, 'channels', 'id')})
45                    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')})
46                s7 = s7_all
47                pass
48        else:
49            s3 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users
                  `.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id
                  ')})
50            s4 = util.do_sql(conn, "SELECT `channels`.* FROM `channels`", {})
51    else:
52        pass
53    return util.add_warnings(outputs)
```

A.29 Kandan Chat Room Command get_channels_id_activities

```
1   def get_channels_id_activities (conn, inputs):
2       util.clear_warnings()
3       outputs = []
4       s0 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users`.`
            username` = :x0 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 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id')})
7           s2 = util.do_sql(conn, "SELECT `channels`.* FROM `channels`", {})
8           if util.has_rows(s2):
9               s5 = util.do_sql(conn, "SELECT `activities`.* FROM `activities`
                    WHERE `activities`.`channel_id` IN :x0", {'x0': util.get_data(s2
                    , 'channels', 'id')})
10              if util.has_rows(s5):
11                  s10 = util.do_sql(conn, "SELECT `users`.* FROM `users`  WHERE `
                        users`.`id` IN :x0", {'x0': util.get_data(s5, 'activities',
                        'user_id')})
12                  s11 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                        users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, '
                        users', 'id')})
13                  s12 = util.do_sql(conn, "SELECT  `channels`.* FROM `channels`
                        WHERE `channels`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
14                  if util.has_rows(s12):
```

```
15                      s13 = util.do_sql(conn, "SELECT  `activities`.* FROM `
                            activities`  WHERE `activities`.`channel_id` = :x0 ORDER
                             BY id LIMIT 1", {'x0': inputs[1]})
16                      outputs.extend(util.get_data(s13, 'activities', 'id'))
17                      outputs.extend(util.get_data(s13, 'activities', 'content'))
18                      outputs.extend(util.get_data(s13, 'activities', 'channel_id'
                            ))
19                      outputs.extend(util.get_data(s13, 'activities', 'user_id'))
20                      s14 = util.do_sql(conn, "SELECT  `activities`.* FROM `
                            activities`  WHERE `activities`.`channel_id` = :x0 ORDER
                             BY id DESC LIMIT 30", {'x0': inputs[1]})
21                      outputs.extend(util.get_data(s14, 'activities', 'id'))
22                      outputs.extend(util.get_data(s14, 'activities', 'content'))
23                      outputs.extend(util.get_data(s14, 'activities', 'channel_id'
                            ))
24                      outputs.extend(util.get_data(s14, 'activities', 'user_id'))
25                      if util.has_rows(s14):
26                          s15 = util.do_sql(conn, "SELECT `users`.* FROM `users`
                                WHERE `users`.`id` IN :x0", {'x0': util.get_data(s14
                                , 'activities', 'user_id')})
27                          outputs.extend(util.get_data(s15, 'users', 'id'))
28                          outputs.extend(util.get_data(s15, 'users', 'email'))
29                          outputs.extend(util.get_data(s15, 'users', 'first_name')
                                )
30                          outputs.extend(util.get_data(s15, 'users', 'last_name'))
31                          outputs.extend(util.get_data(s15, 'users', 'username'))
32                      else:
33                          pass
34                  else:
35                      pass
36              else:
37                  s6 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `
                        users`.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, '
                        users', 'id')})
38                  s7 = util.do_sql(conn, "SELECT  `channels`.* FROM `channels`
                        WHERE `channels`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
39                  if util.has_rows(s7):
40                      s8 = util.do_sql(conn, "SELECT  `activities`.* FROM `
                            activities`  WHERE `activities`.`channel_id` = :x0 ORDER
                             BY id LIMIT 1", {'x0': inputs[1]})
41                      s9 = util.do_sql(conn, "SELECT  `activities`.* FROM `
                            activities`  WHERE `activities`.`channel_id` = :x0 ORDER
                             BY id DESC LIMIT 30", {'x0': inputs[1]})
42                  else:
43                      pass
44          else:
45              s3 = util.do_sql(conn, "SELECT  `users`.* FROM `users`  WHERE `users
                    `.`id` = :x0 LIMIT 1", {'x0': util.get_one_data(s0, 'users', 'id
                    ')})
46              s4 = util.do_sql(conn, "SELECT  `channels`.* FROM `channels`  WHERE
                    `channels`.`id` = :x0 LIMIT 1", {'x0': inputs[1]})
47      else:
48          pass
49      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", {})
```

```
8       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)
```