Towards a Method for Reducing the Test Suites of Database Applications

Gregory M. Kapfhammer
Department of Computer Science, Allegheny College

The Prevalence of Database Applications

Electronic journals, scientific data repositories, and e-commerce systems

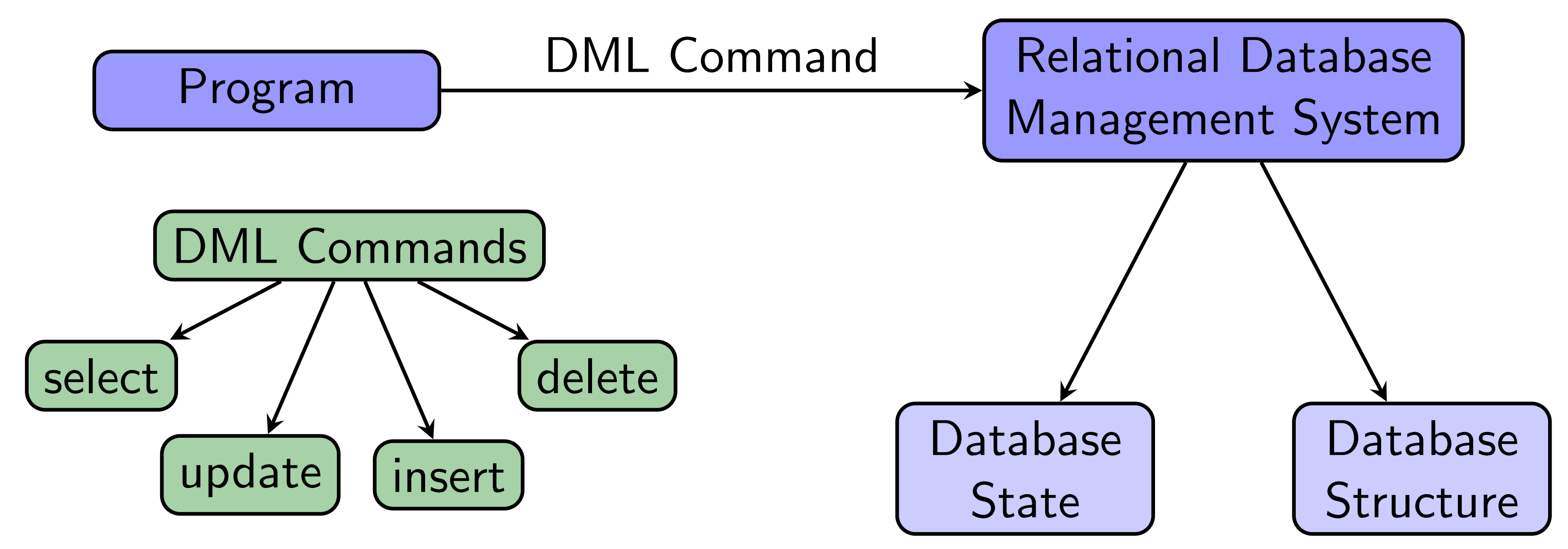
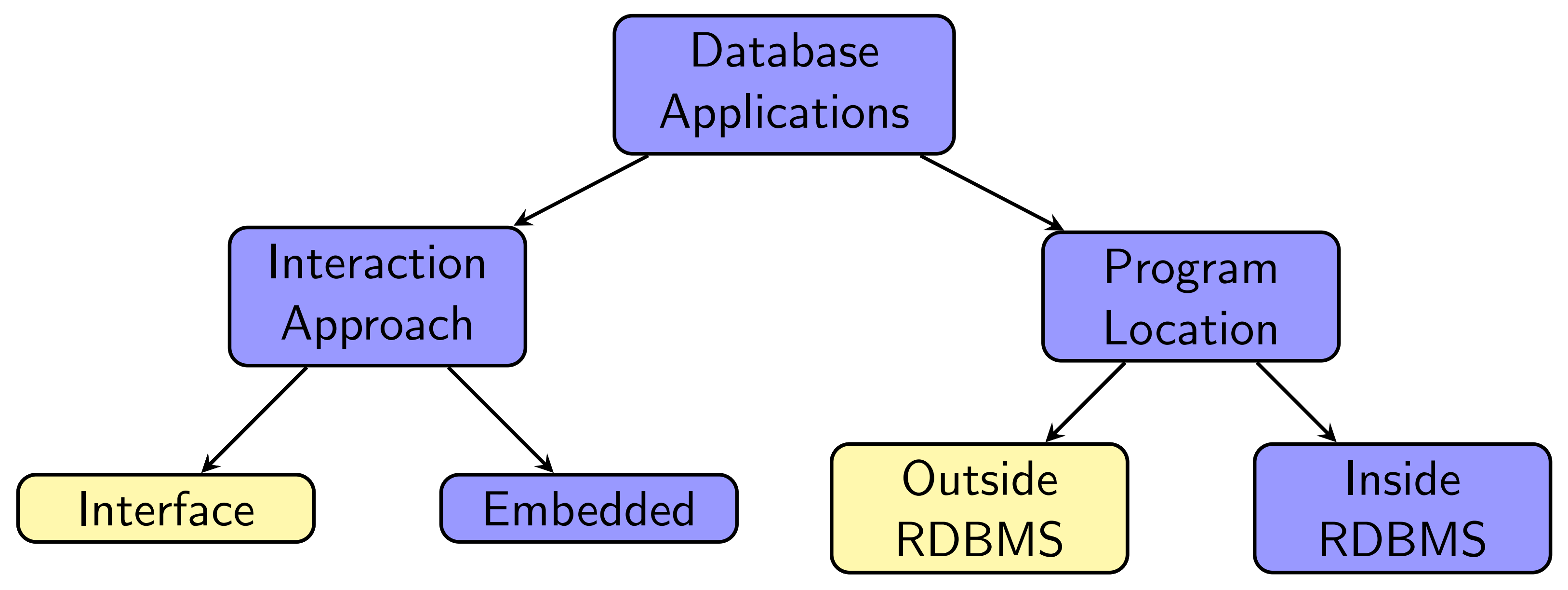


Figure: Common Architecture of Many Real-World Applications That Interact with a Relational Database.

- Silberschatz et al. observe that "practically all use of databases occurs from within application programs" [Data. Sys. Conc. 2010]
- Database applications rapidly evolve as changes are made to both the program and the database's state and structure

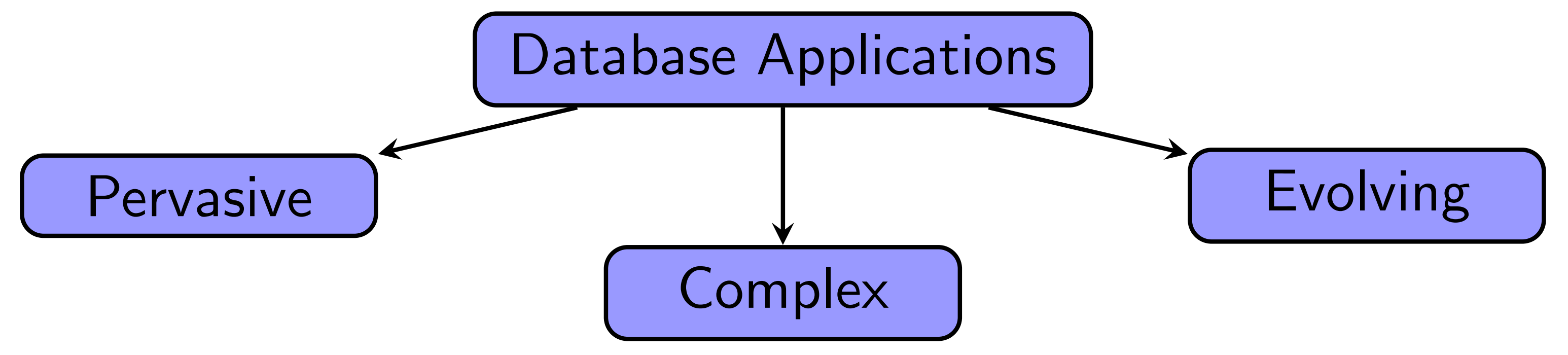
Types of Database Applications



Java application that submits SQL strings to MySQL using JDBC

Figure: Categorizing Database Applications — the Presented Method Focuses on the Highlight Type of Application.

The Role of Test Suite Reduction

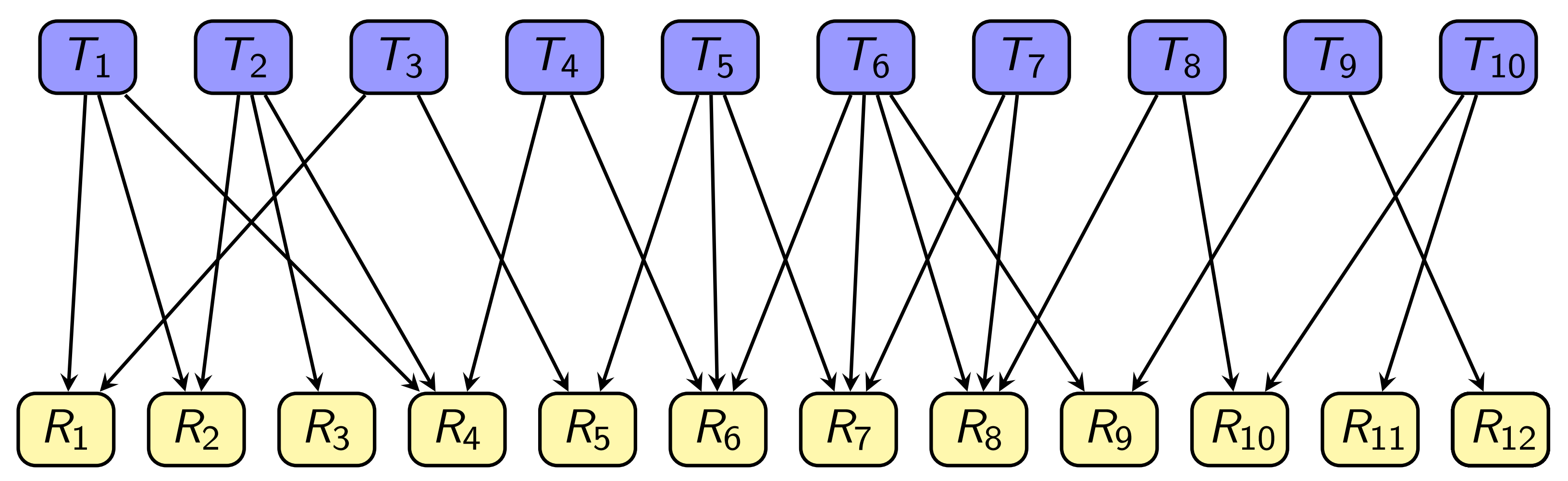


By removing redundant test cases, **test suite reduction** supports the efficient modification of database applications

Figure: Test Suite Reduction Aims to Improve the Efficiency of Testing Database Applications.

Database-Aware Test Suite Reduction

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$



Requirements Set $R = \{R_1, R_2, \ldots, R_{11}, R_{12}\}$

Granularity varies from database to attribute value

See [Kapfhammer and Soffa, ISEC 2008] for more details

Overlap-aware test suite reduction greedily removes redundant tests

Figure: The Process of Test Suite Reduction for Database Applications.

Case Study Applications

Name	Classes	Methods	NCSS	Per
Reminder (RM)	9	55.0	548.0	*Program*
		6.11	60.89	*Class*
			9.96	*Method*
FindFile (FF)	5	49.0	558.0	*Program*
		9.8	111.6	*Class*
			11.39	*Method*
Pithy (PI)	11	73.0	579.0	*Program*
		6.64	52.64	*Class*
			7.93	*Method*
StudentTracker (ST)	9	72.0	620.0	*Program*
		8.0	68.89	*Class*
			8.61	*Method*
TransactionManager (TM)	6	87.0	748.0	*Program*
		14.5	124.67	*Class*
			8.6	*Method*
GradeBook (GB)	10	147.0	1455.0	*Program*
		14.7	145.5	*Class*
			9.9	*Method*

Table: High-Level Description of the Case Study Applications Used in the Empirical Study.

Experimental Results

A - $|T|$	Rel.	Attrib.	Rec.	Attrib. Val.	All
RM - 13	(7, .46)	(7, .46)	(10, .3)	(9, .31)	(8.25, .37)
FF - 16	(7, .56)	(7, .56)	(11, .31)	(11, .31)	(9, .44)
PI - 15	(6, .6)	(6, .6)	(8, .7)	(7, .53)	(6.75, .55)
ST - 25	(5, .80)	(5, .76)	(11, .56)	(10, .6)	(7.75, .69)
TM - 27	(14, .48)	(14, .48)	(15, .45)	(14, .48)	(14.25, .47)
GB - 51	(33, .35)	(33, .35)	(33, .35)	(32, .37)	(32.75, .36)
All - 24.5	(12, .51)	(12.17, .5)	(14.67, .4)	(13.83, .44)	

Table: The Reduction in Test Suite Size for the Database Applications with ($|T'|$, RFFS(T, T')) for All Data Points.

- RFFS$(T, T') = (|T| - |T'|) \div |T|$
- ST has the best RFFS (.69 avg) and GB has the worst (.36 avg)
- Across all of the applications, RFFS was .51 on average at the relation level and .44 on average at the attribute value level
- RFFS drops from .50 to .40 when the reducer analyzes at the record level instead of the attribute level
- RFFS climbs to .44 from .40 with attribute value requirements

Application	Relation	Attribute	Record	Attribute Value	All
RM	.07	.07	.04	.05	.07
FF	.13	.13	.08	.08	.11
PI	.29	.29	.15	.18	.23
ST	.19	.18	.13	.13	.16
TM	.23	.23	.19	.22	.22
GB	.78	.78	.78	.78	.78
All	.28	.28	.23	.24	

Table: The Reduction in Test Suite Time for the Database Applications with RFFT(T, T') for All Data Points.

- RFFT$(T, T') = (time(T) - time(T')) \div time(T)$
- GB has the highest RFFT value because it contains redundant tests that restart the database and are thus very costly to run
- Except for GB, the RFFT values are lower than those for RFFS
- RFFS was .28 on average at the relation level and .24 on average at the attribute value level, across all of the applications
- When the reducer analyzes at the record level instead of the attribute value level, RFFS decreases from .28 to .23
- With attribute value requirements RFFS increases to .24 from .23

Future Work

- Use larger and more varied applications in follow-on experiments
- Investigate the fault-detection effectiveness of T and T'
- Focus on affiliated testing tasks (e.g., test data generation)