重新思考Web 场景下的事务抽象与SQL优化问题

Zhaoguo Wang

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

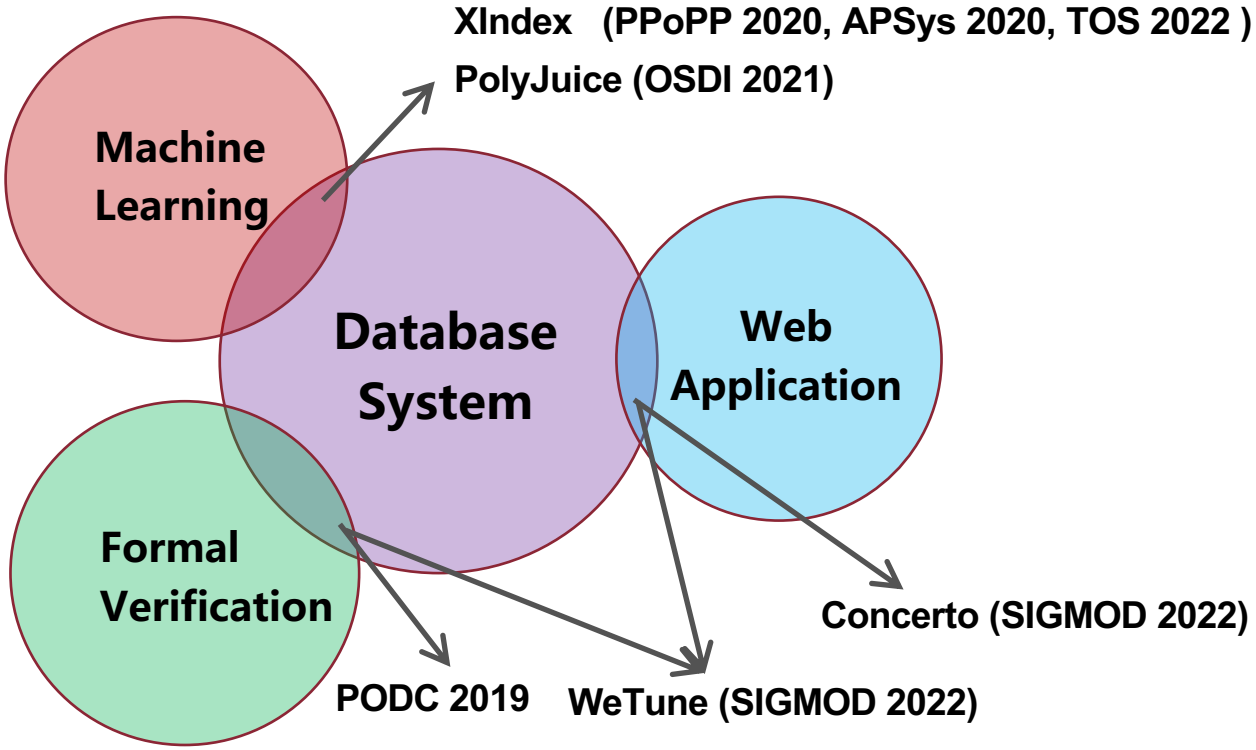
Recent 3-4 Years

Transaction Processing

Concurrent Index

Consensus

SQL Optimization



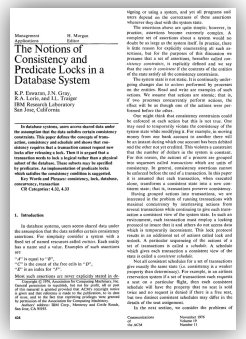
Database has a long history

Transaction

1960s

IDS
Integrated Data Store

Charles Bachman
(1973 Turing Award)


The Notions of Consistency and Predicate Locks in a Database System

K.P. Eswaran, J.N. Gray, R.A. Lorie, I.L. Traiger
IBM Research Laboratory
San Jose, California

TXN Concept (1976)


Principles of Transaction-Oriented Database Recovery

THEO HAERDER
Fachbereich Informatik, University of Kaiserslautern, West Germany

ANDREAS REUTER
IBM Research Laboratory, San Jose, California 95150

ACI → ACID (1983)


A Critique of ANSI SQL Isolation Levels

Weak isolation (1995)

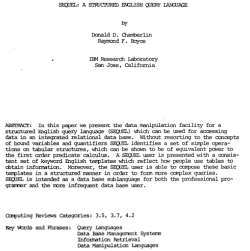
...

SQL


A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Relational Model (1970)

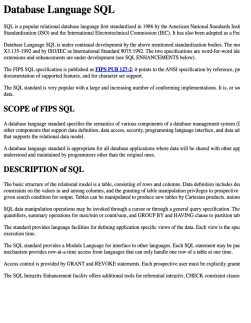

SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

SEQUEL (1974)

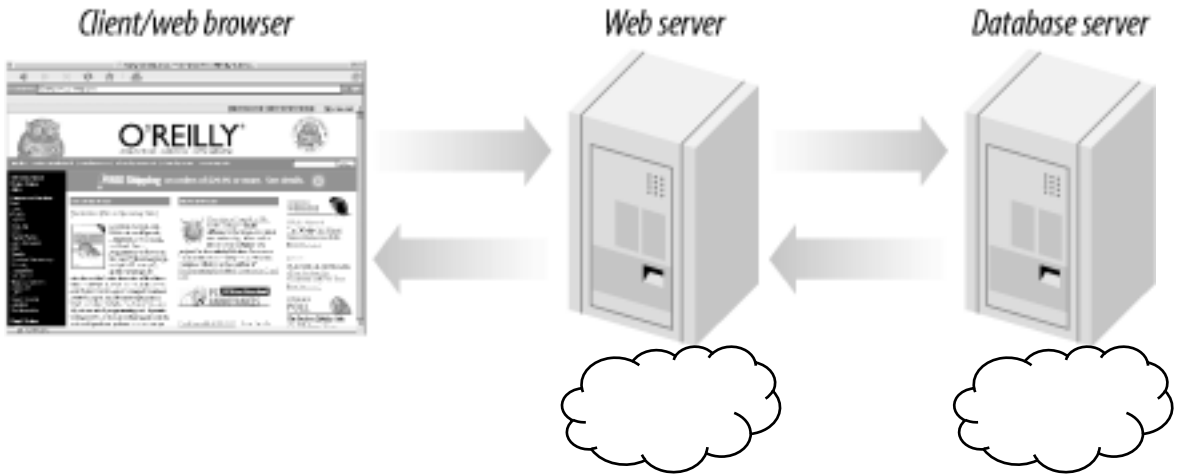

Database Language SQL

ISO Standard (1986)

...

The most popular database application: web applications

- Web applications are already integrated in our daily life: socialization, entertainment, work, etc.
- They unanimously use one or more database systems to manage and access their data.



Web apps are constantly evolving

Web dictionary
(1994)

Search engine
(1998)

Social network
(2004)

Mobile apps
(2008)

...

A simple question

- Does the decade-old database transaction abstraction and SQL optimization methods still fit web applications today?

First paper
(1976)

ACI → ACID
(1983)

Weak isolation
(1995)

?

Relational Model
(1970)

SEQUEL
(1970)

ISO Standard
(1986)

Web dictionary
(1994)

Search engine
(1998)

Social network
(2004)

Mobile apps
(2008)

What is the answer?

Existing works: Query Abstraction (SQL v.s. NOSQL)

This Talks

- **Transaction Abstraction**
- **Query Optimization**

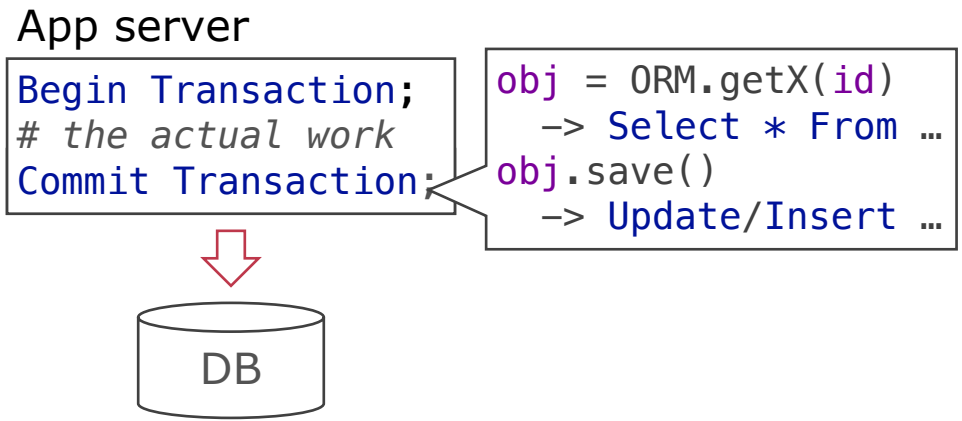
Transaction Abstraction

Ad Hoc Transactions in Web Applications:
The Good, the Bad, and the Ugly

Chuzhe Tang, **Zhaoguo Wang,** Xiaodong Zhang, Qianmian Yu
Binyu Zang, Haibing Guan, Haibo Chen

How do applications today use transaction?

- Intuitively, database transactions.



How do applications today use transaction?

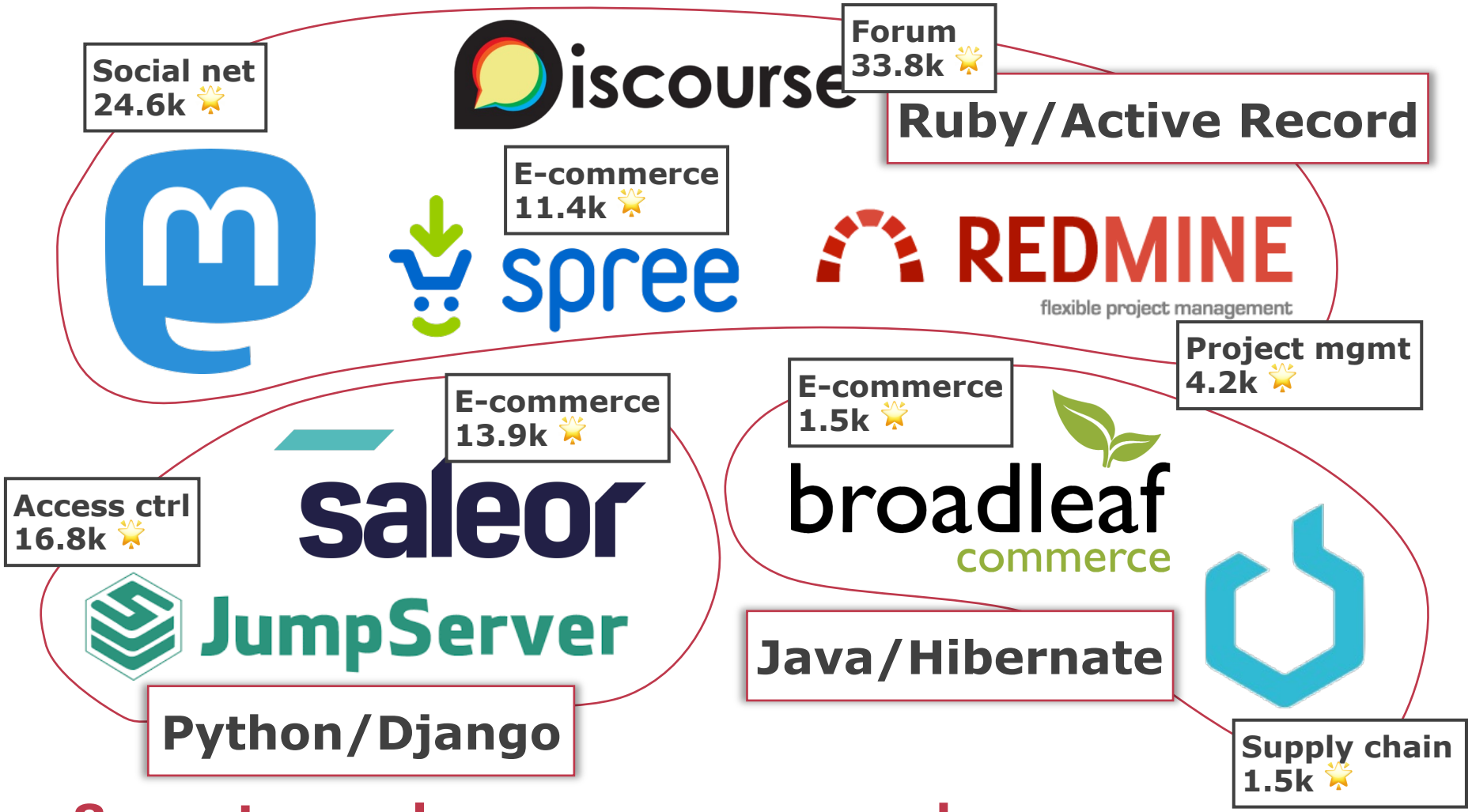
- Bailis et al. identified another application-level approach: invariant validation.
  - Developers specify invariants; ORMs validate them.

App server

```
class Account:  # => Accounts table
  string email  # => email column
  validates :email, uniqueness: true
```

```
acc = new Account(email: “a@b.com”)
acc.save()
```

```
cnt = Select count(*) From Accounts
      Where email=“a@b.com”
if cnt == 0:
  Insert Into Accounts (email) Values (“a@b.com”)
```



8 most popular open-source web apps:

Different types, languages and ORM frameworks

Ad hoc transaction

They are the “transactions” coordinated by ad hoc constructs (e.g., locks) employed by app developers.

App server (add-cart API)

```
lock(cart_id)
# perform business logic
# use ORM to access DB
unlock(cart_id)
```

Server-side lock table

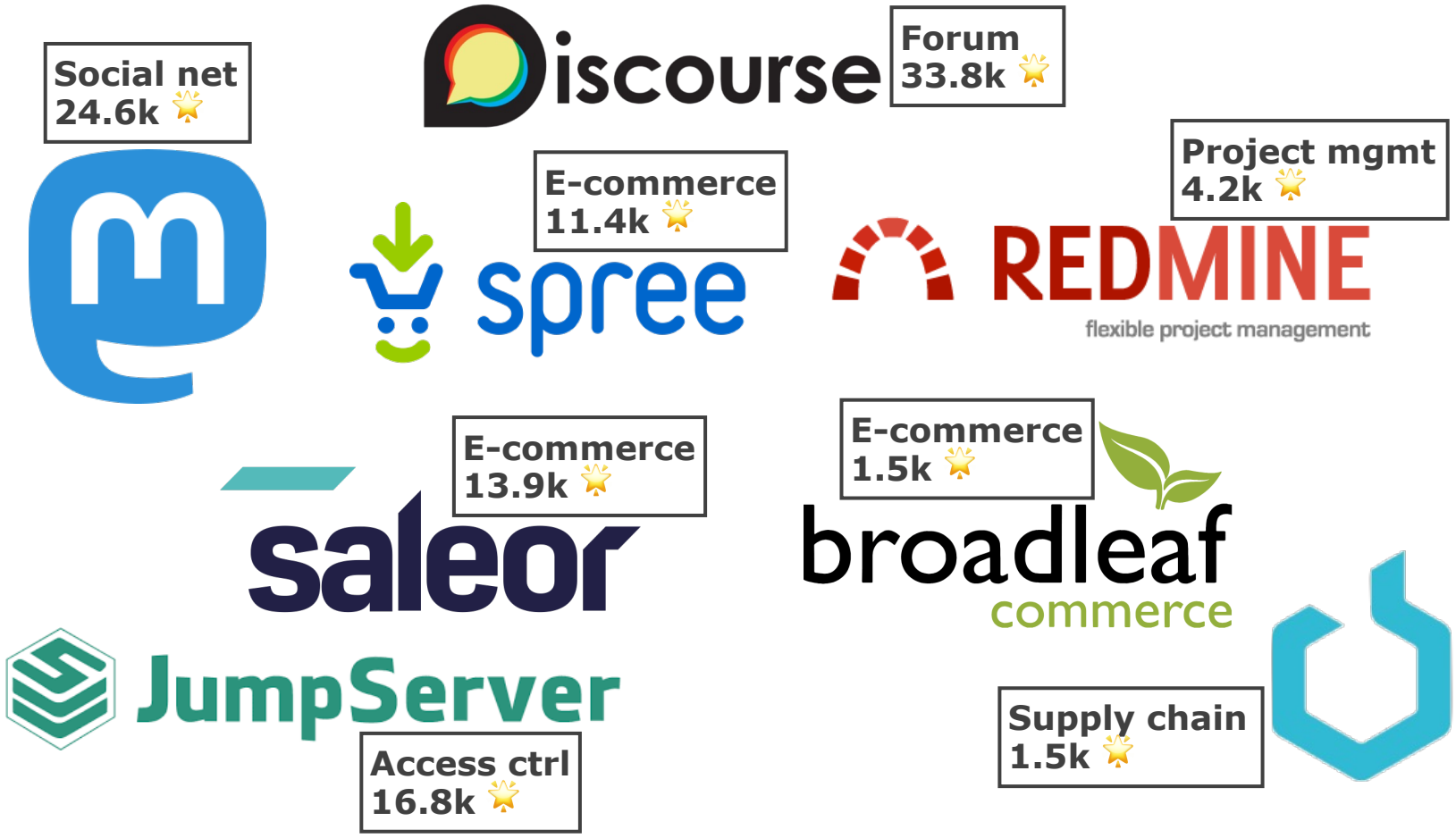
cart	locked
1	true
2	false

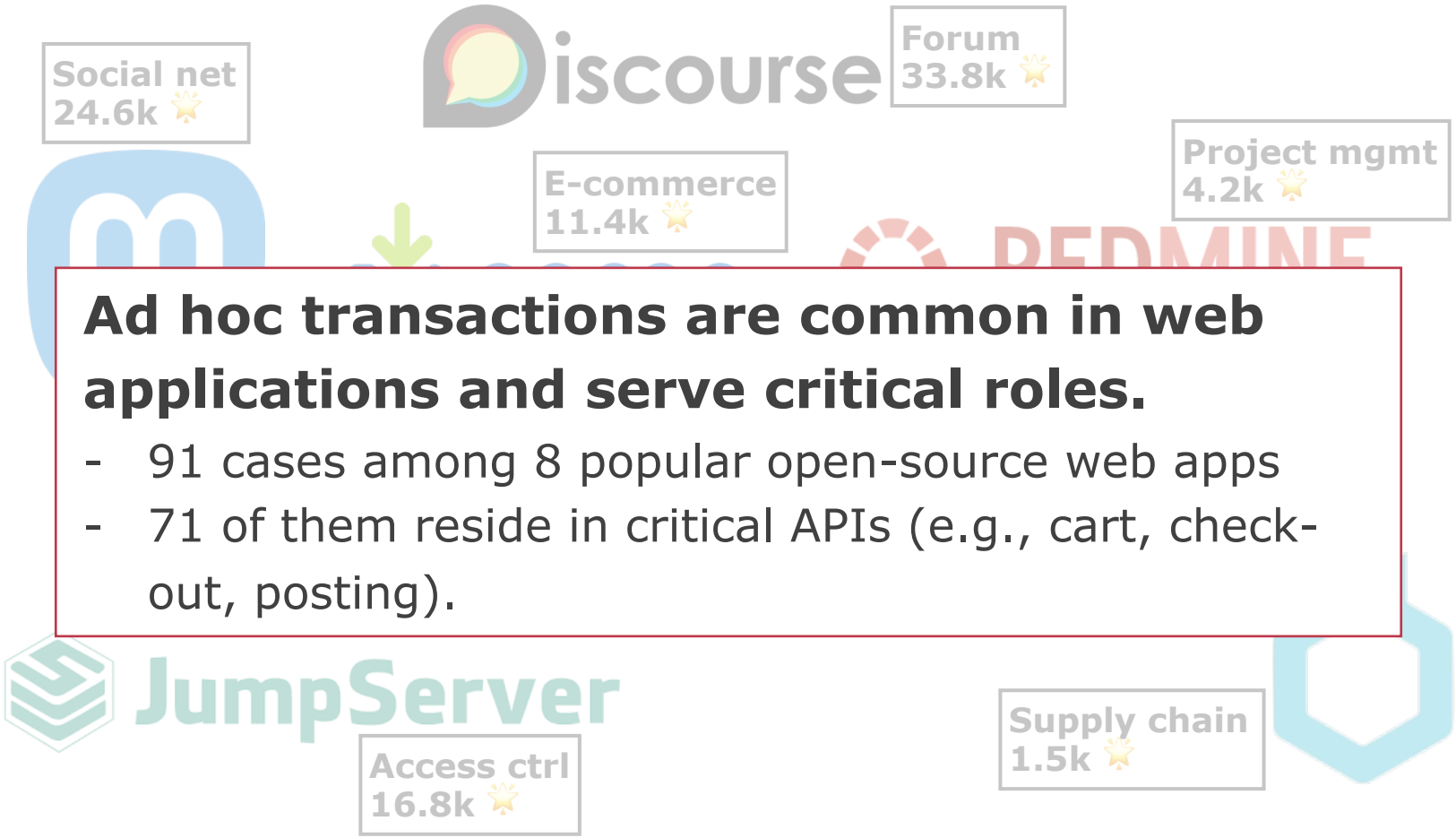
Plain `Select`/`Update`/`Insert`/`Delete`
(without DB transactions)

Ad hoc transactions represent a third approach

	DB transactions	Invariant validation	Ad hoc transactions
WHAT is protected?	Business logic snippets	Invariants on DB rows	Business logic snippets
WHO conduct the protection?	DB CC	ORMs	Developers

What is the state of the practice?


Social net
24.6k
Forum
33.8k
Discourse
Project mgmt
4.2k
E-commerce
11.4k
spree
REDMINE
flexible project management
E-commerce
13.9k
saleor
E-commerce
1.5k
broadleaf
commerce
JumpServer
Access ctrl
16.8k
Supply chain
1.5k

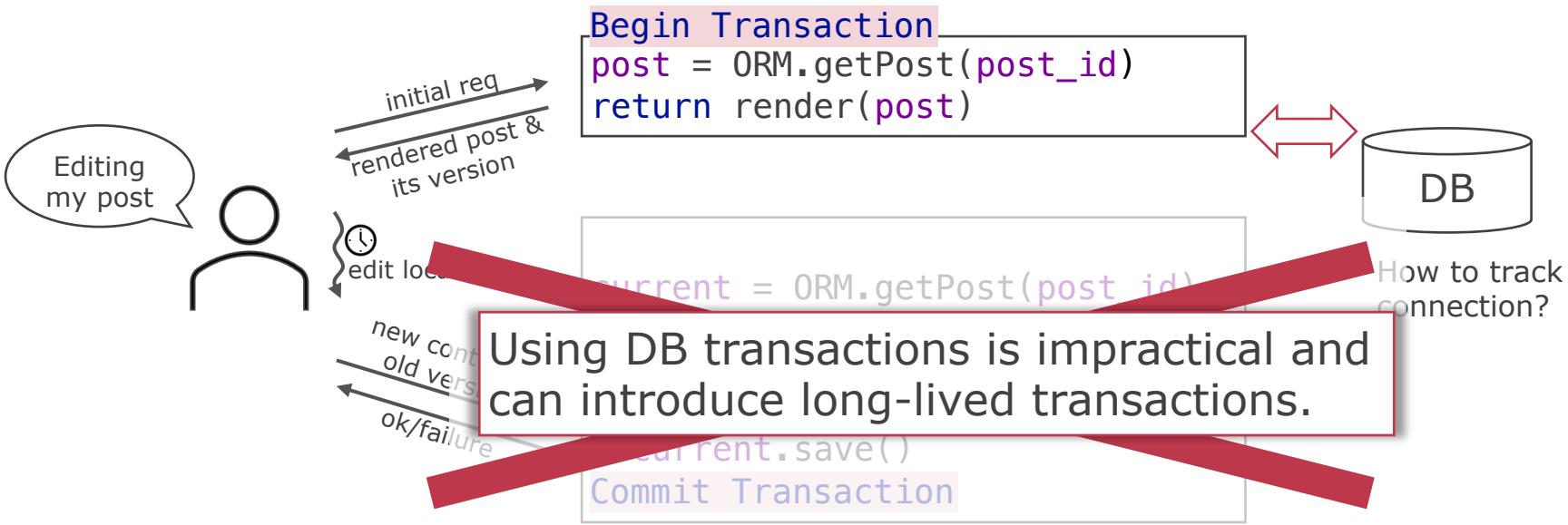

Social net
24.6k
Forum
33.8k
E-commerce
11.4k
Project mgmt
4.2k
Ad hoc transactions are common in web applications and serve critical roles.
- 91 cases among 8 popular open-source web apps
- 71 of them reside in critical APIs (e.g., cart, check-out, posting).
JumpServer
Access ctrl
16.8k
Supply chain
1.5k

Answer the following questions.

- **How do ad hoc transactions look like?**
- **Are they correct?**
- **Do they perform well?**

Ad hoc transactions have diverse semantics

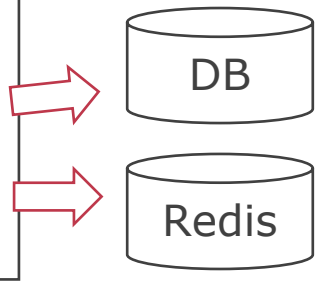
- Their coordination can span many requests.
  - 10/91 cases coordinate through multiple requests.



Ad hoc transactions have diverse semantics

- They can also coordinate non-DB operations.
  - 8/91 cases handle non-DB operations.

```
lock(post_id)
post = new Post(...)
post.save()
REDIS.add_to_set(
  “timeline:”+ follower_id, post_id)
unlock(post_id)
```



Post table

id	content
14	foo
…	…

```
“timeline:xx”:
   post_id          (sort order) add_time
  (14,  21/9/20 23:59:59)
  (...)

“timeline:yy”: ...
```

DB transactions (almost) cannot coordinate external storage systems (e.g., Redis, S3).

Ad hoc transactions have diverse implementations

- They use either locks or validation procedures for coordination.
- For locks, there are 7 different implementations among 8 applications.
  - 2 implementations reuse existing locking facilities.
  - 2 implementations store lock info in Redis.
  - 1 implementations store lock info in DB tables.
  - 2 implementations store lock info in application runtime containers (e.g., Java’s ConcurrentHashMap).

Ad hoc transactions have diverse implementations

- For validation procedures, there are also 2 categories.
  - One is generated by the ORM according to annotations.

```
class Person:
  string name
  @Version
  int ver
```

```
john = ORM.getPerson(...)
john.setName("Bob")
john.save()
```

```
Update Persons Set name="Bob", ver=john.ver+1
Where id=john.id And ver=john.ver
```

The DB system ensures version check and update happen atomically.

  - One is implemented from scratch by developers.
    - Developers need to ensure the check-and-update atomicity.
    - E.g., the multi-request example shown before.

Ad hoc transactions handle failures differently

- Developers do not handle deadlocks, yet we didn’t find potential deadlocks.
  - Probably due to the reduced number of locks.
- In 6 cases, developers write rollback/repair procedures to handle conflicts.
- One application has periodic DB checks (like `fsck`) to fix inconsistency.
  - E.g., post referring an absent image.

Are ad hoc transactions correct?

- 69 correctness issues are found in 53 cases.
  - Some cases suffer from multiple issues.
  - 33 cases' issues are confirmed by developers.
- We consider 28 of them severe.

App.	Known severe consequences	Cases
Discourse	Overwritten post contents, page rendering failure, excessive notifications.	6
Mastodon	Showing deleted posts, corrupted account info., incorrect polls.	4
Spree	Overcharging, inconsistent stock level, inconsistent order status, selling discontinued products.	9
Broadleaf	Promotion overuse, inconsistent stock level, inconsistent order status, overselling.	6
Saleor	Overcharging.	3

Majority of issues stem from wrong locking/validation primitives

- 36/65 lock-based ad hoc transactions wrongly implement or use locking primitives.
- 11/26 validation-based ad hoc transactions failed to ensure check-and-update atomicity.

```
ORM.transaction:
  ok = MiniSql.query(
    Update Posts Set version=version+1
    Where id=
  if not ok:
    ORM.abort_transaction()
  # perform updates here
```

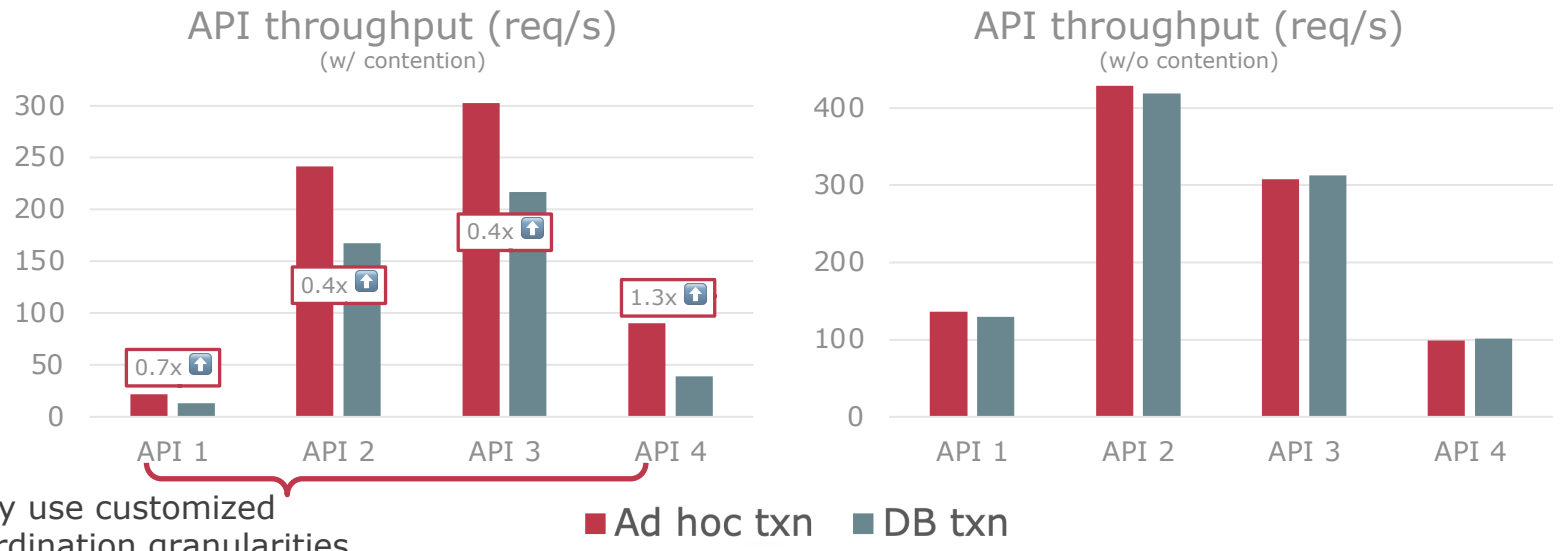
MiniSql is independent of the ORM, thus issuing Update outside of the DB transaction.

Developers sometimes wrongly employ ad hoc transactions

- 16 issues are caused by incorrect scope.
  - Developers might omit critical operations from coordination in existing ad hoc transactions. (11 cases)
  - Developers might forget to employ ad hoc transactions for conflicting procedures. (5 cases)
- 4 issues are caused by incorrect failure handling.
  - E.g., crash during ad hoc transactions introduce invalid intermediate states that cause user blocking after reboot.

Do ad hoc transactions perform well?

- We deployed the applications and evaluated a subset of APIs with synthetic workloads.
  - In comparison with codebase modified to use DB transactions.



They use customized coordination granularities

What does it imply?

- Why do developers not use DB transactions?
  - Lacking important functionalities/properties?
  - Need better integration with applications?
  - Maybe applications are fine with relaxing ACID semantics?
- What should we do?
  - Further investigation why they use ad hoc transactions.
  - Explore new concurrency abstraction to better suit applications today.
  - Build tools/Sync. Primitives to improve existing web applications.

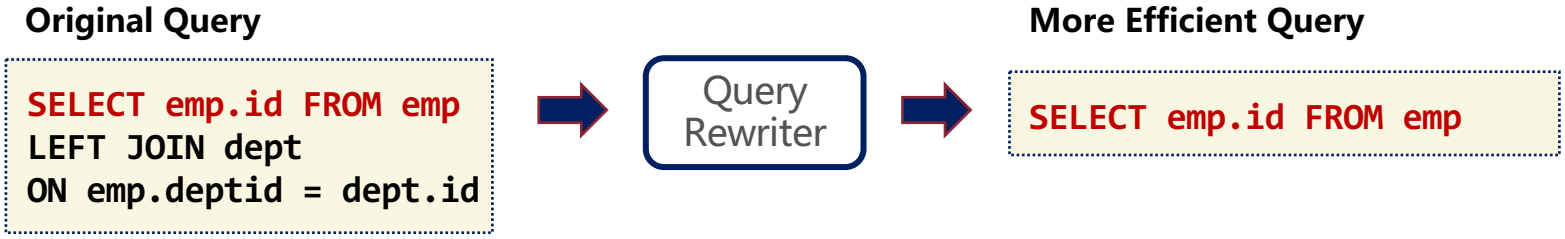
Query Optimization

WeTune: Automatic Discovery and Verification of Query Rewrite Rules

Zhaoguo Wang, Zhou Zhou, Yicun Yang, Haoran Ding, Gansen Hu, Ding Ding, Chuzhe Tang, Haibo Chen, Jinyang Li

Background: Query Rewrite

Query rewriting is an important step in query optimization.



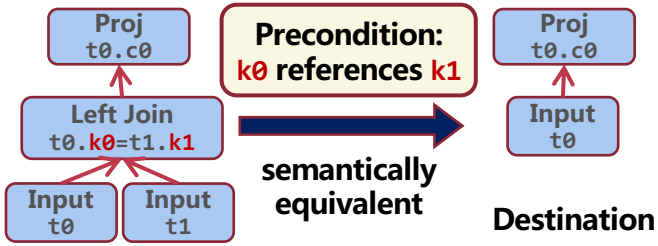
Query Rewrite in Apache Calcite[1]

1. Begoli et al. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. SIGMOD '18.

Rule-Based Query Rewrite

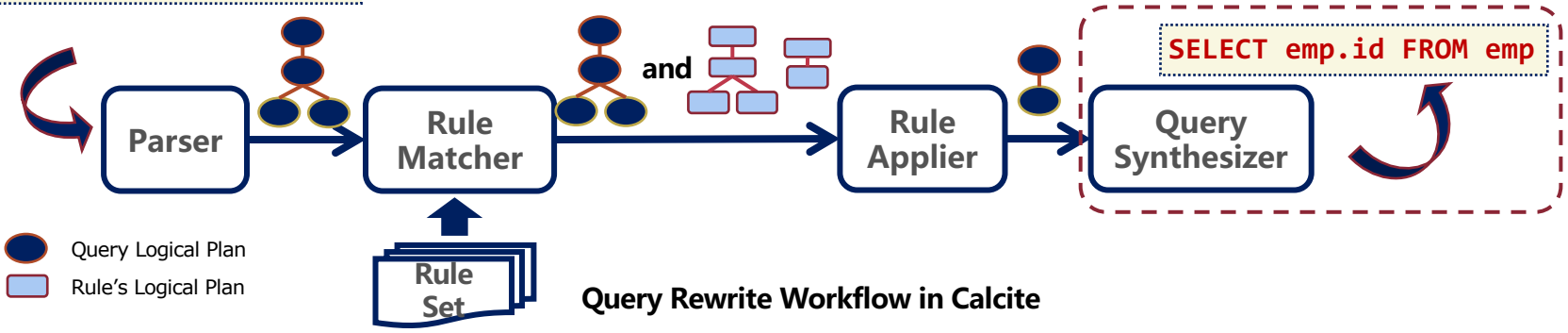
Normally, a rule consists of a pair of logical query plans.

- Source logical plan: match the query.
- Destination logical plan: rewrite the query.



A Rule Example

```
SELECT emp.id FROM emp
LEFT JOIN dept
ON emp.deptid = dept.id
```

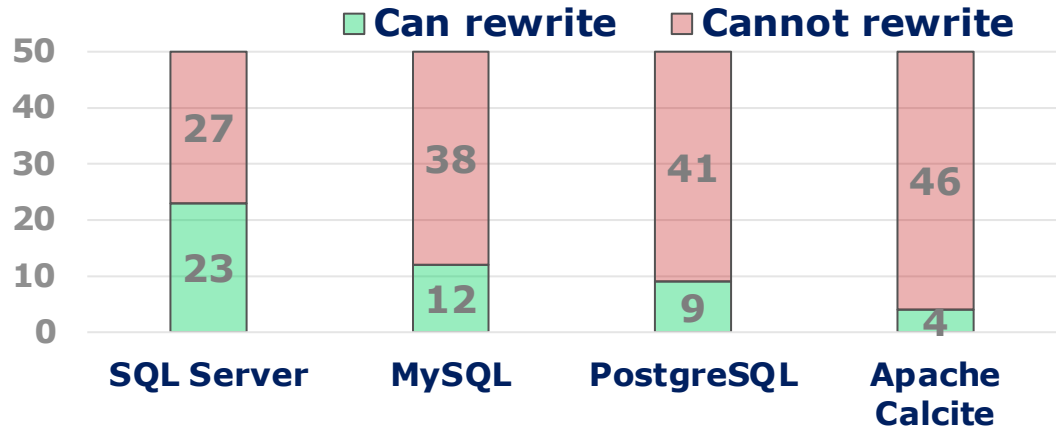


Query Rewrite Workflow in Calcite

Query Rewriting in Web Applications

Existing accumulated rules are far from sufficient to rewrite web application queries.

- Miss many rewrite opportunities.
- Survey on 50 GitHub query performance issues:



Existing Rules Fail with Web App Queries

Web application queries might be counter-intuitive.

- Pervasively use object-relational mapping (ORM) framework to generate queries.

Counter-intuitive query patterns might not match existing rules.

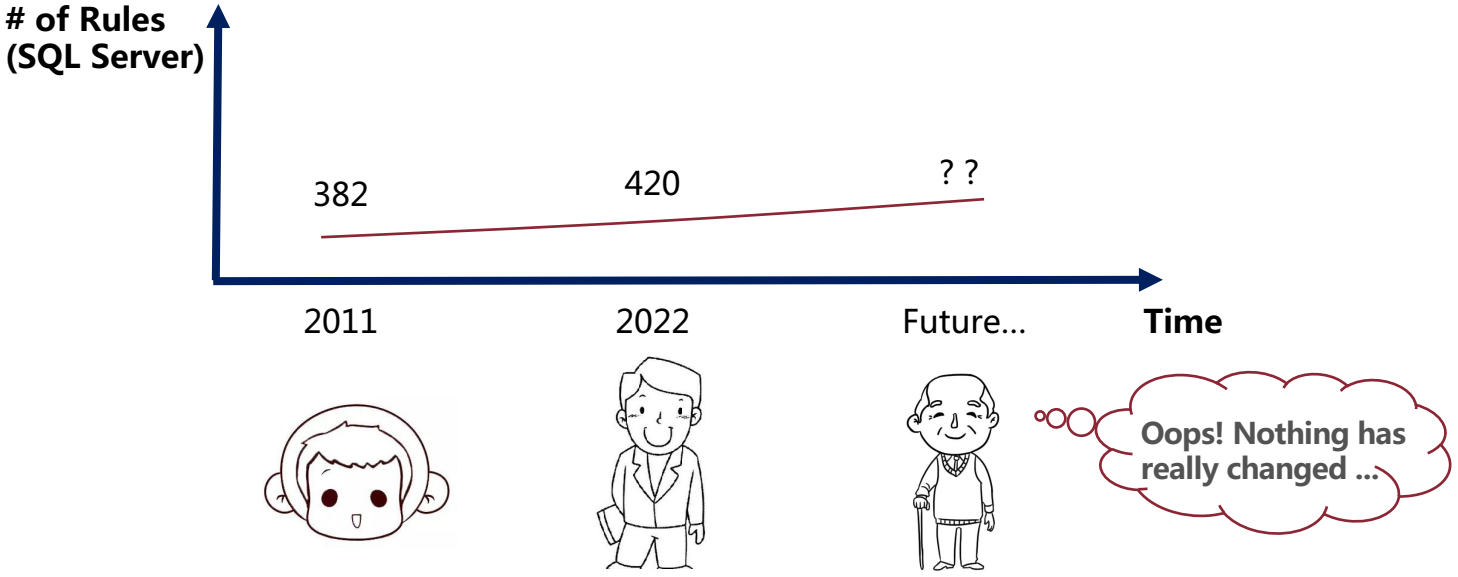
```
items = labels.with_label
  (label_names, params[:sort])
items_projs = projects(items)
......
label_ids = LabelsFinder.new(
  current_user,
  project_ids: items_projs).select(:id)
items = items.where(items: {id: label_ids})
```

```
SELECT * FROM labels
WHERE id IN (
  SELECT id FROM labels
  WHERE id IN (
    SELECT id FROM labels
    WHERE proj_id = 10
  ) ORDER BY title ASC
)
```

Drawback of Existing Practice

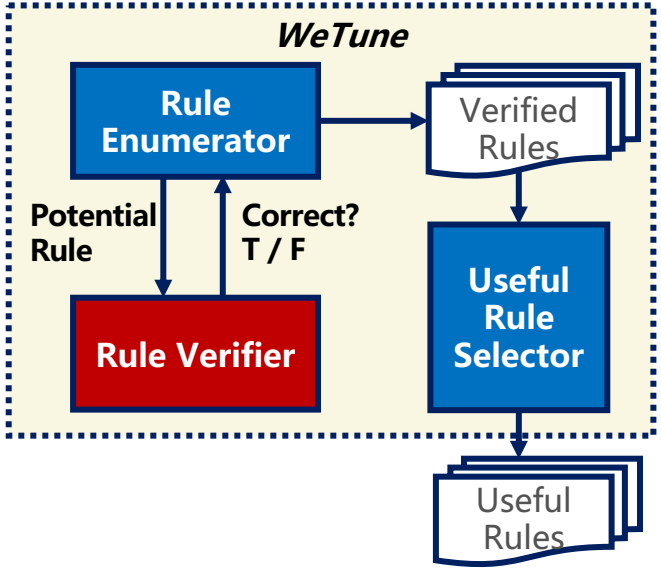
Rules in existing systems are empirically crafted by human's manual efforts.

- Take decades to accumulate rules.



Basic Idea: Automatically Discover Rules

- Enumerating rules by brute-force.
- Ensure correctness of rules by verification.



Challenges

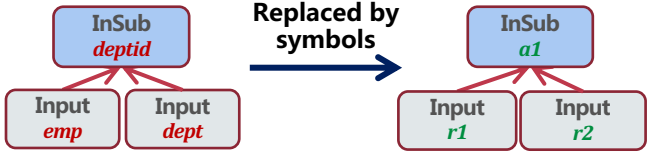
How to efficiently enumerate rewrite rules?

How to verify correctness of rules?

Defining Rules

Model a rewrite rule: $<Q_{src}, Q_{dest}, C>$.

- Q_{src}, Q_{dest}: source/destination **plan template.**



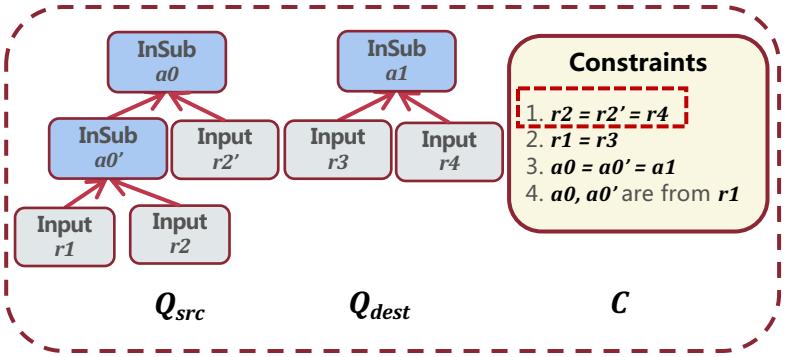
Concrete plan

```
SELECT * FROM emp
WHERE deptid IN
 (SELECT deptid FROM
dept)
```

Plan template

```
SELECT * FROM r1
WHERE a1 IN
 (SELECT a1 FROM r2)
```

- C: the precondition of the rule.
  - A set of **constraints** over symbols.



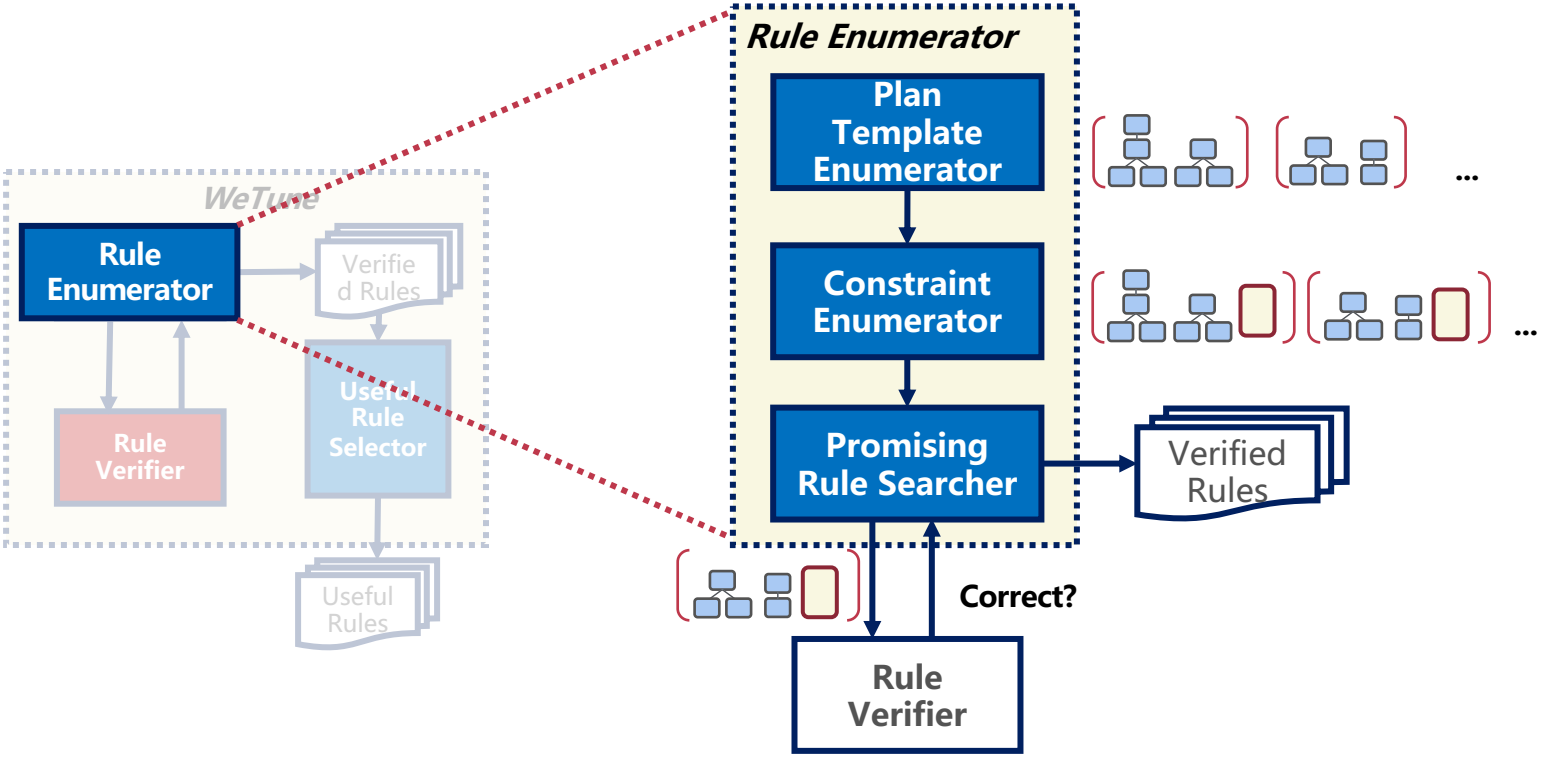
An new rule found by WeTune

```
Q1: SELECT * FROM emp
    WHERE emp.deptid IN (SELECT dept.id FROM dept)
      AND emp.deptid IN (SELECT dept.id FROM dept)
```

```
Q2: SELECT * FROM emp
    WHERE emp.deptid IN (SELECT dept.id FROM dept)
```

A correct rule means C => $(Q_{src} \equiv Q_{dest})$.

Rule Enumerator Overview



Built-in Rule Verifier Overview

What does a rewrite rule look like?

$$\langle Q_{src}, Q_{dest}, C \rangle \xrightarrow{\textit{rewrite}} \text{Optimized SQL}$$

What is the correctness of a rule?

$$C \rightarrow (Q_{src} \equiv Q_{dest})$$

If the constraints in C are satisfied, then Q_{src} and Q_{dest} are equivalent → Proving the equivalence of two SQL statements.

Why cannot use existing SQL solvers?

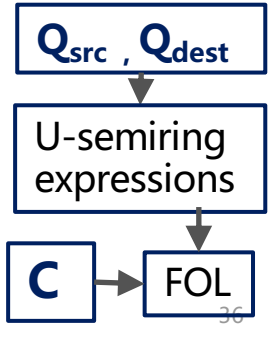
Cosset	Only support set semantic
UDP	Do not support Outer Join and NULL
SPES	Q_{src} and Q_{dest} must have the same inputs.

Our Solution: 10X powerful

Step 1. Convert Q_{src} and Q_{dest} into U-expressions

Step 2. Convert $C \rightarrow (Q_{src} \equiv Q_{dest})$ into first order logic formulas.

Step 3. Use SMT solver to solve the FOL automatically.



Evaluation

Q1. How many new useful rules can WeTune discover?

Q2. How effective are the discovered useful rules?

Setup:

- 8518 queries collected from 20 open-source web applications on GitHub.
- Evaluate with SQL Server 2019.

Evaluation

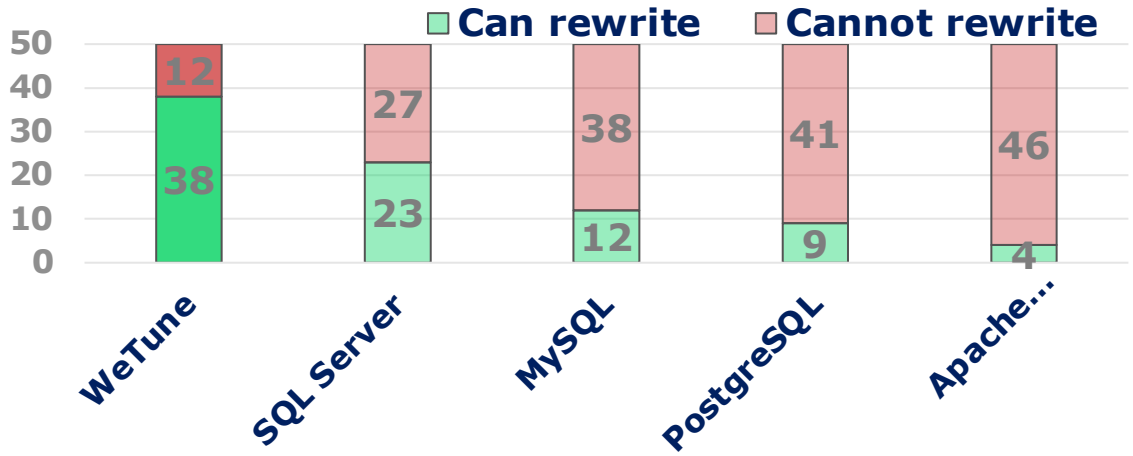
Q1. How many new useful rules can WeTune discover?

- Discover **35** useful rules based on 8518 real-world queries.
- **9** are missing in SQL Server and **22** are missing in Apache Calcite.

Evaluation

Q2. How effective are the discovered useful rules?

- Rewrite **674** of 8518 queries, SQL Server misses **247** rewrites.
  - **13%** achieve more than **90%** latency reduction.
- Fix **38** of 50 GitHub performance issues.



Conclusion

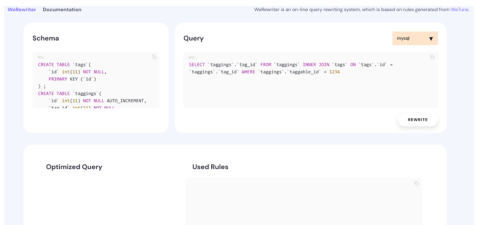
Does the decade-old database transaction abstraction and SQL optimization methods still fit web applications today?

Ad hoc transactions are a common approach to concurrency control in web applications.

WeTune is a tool that automatically discovers query rewrite rules.

https://ipads.se.sjtu.edu.cn:1312/opensource/wetune

https://ipads.se.sjtu.edu.cn/werewriter-demo/home



Questions

Thank You!