Talking to the Database in a Semantically Rich Way:

A New Approach to Resolve Object-Relational Impedance Mismatch.

Henrietta Dombrovskaya
Senior Database Architect
Enova
Chicago IL

hdombrovskaya@enova.com
hdombrovskaya.wordpress.com

enova

About the speaker

- Graduated from the University of Saint Petersburg in 1985 with Applied Math major
- At that time there was no CS major, there was no SE department and there was no AIS department, although most of the people were there
- PhD in CS in 1995
- Worked in different industries and Government bodies, including the City of Chicago Mayors Office, New York Department of Education, Pepsi Americas, Chicago Board of Options…

… what is so special about Enova?

One of the oldest trades..

What is Enova doing?..

enova™

What is object-relational impedance mismatch and why it is bad?

Why should we care?

It's all about application performance!

enova

What contributes to database application performance?

Why to use databases?

Because…

*The DBMS is **specialized software** designed to manage data in the **most efficient way**.*

Nevertheless, the most common complaint of application developers is ….

THE DATABASE IS

SLOW

WHY???

Database developer:

database is always fast,

people just don’t know how to write queries!

Application developer:

application is perfect,

until it hits a database…

enova

May be, we do not have enough hardware?

Our US master PG database runs on

80 thread processors

2.4GHz

512 Gb RAM – almost completely used by disk cache

1066MHz (responses from RAM are 0.9 ns)

I/O 4Gb/sec with avg response time 3ms

I/O utilization: 40%

Even with the best hardware available we can make it only **twice** faster

Current cost**:** **20K** (commodity)

Next – **100K** – somewhat faster (non-commodity)

Next - **1,000K** - twice faster (mainframe)

enova

Now - let's see, *what* is slow…

Where do we *usually* start to look, if we want to see, why the database performance is bad?...

enova

pgBadger Reports

Longest-running queries (slowest queries)

Most frequently running queries

Queries, which take up the most time (top offenders).

enova

Based on these reports - is a database really slow?

```
SELECT * FROM loans WHERE id=?
```

executions: 8,500,000

avg time: <10ms

total execution time about 2.5 hours!

enova

May be, we need this?...

May be, that's how many times this query should be executed?...

Well…

account home controller - 50,000 times during the day

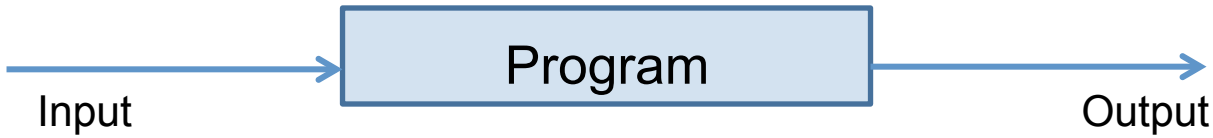
some application controllers: over 1,000 database calls *for each* screen refresh.

How could this possibly be happening?!

Let’s take a step back…

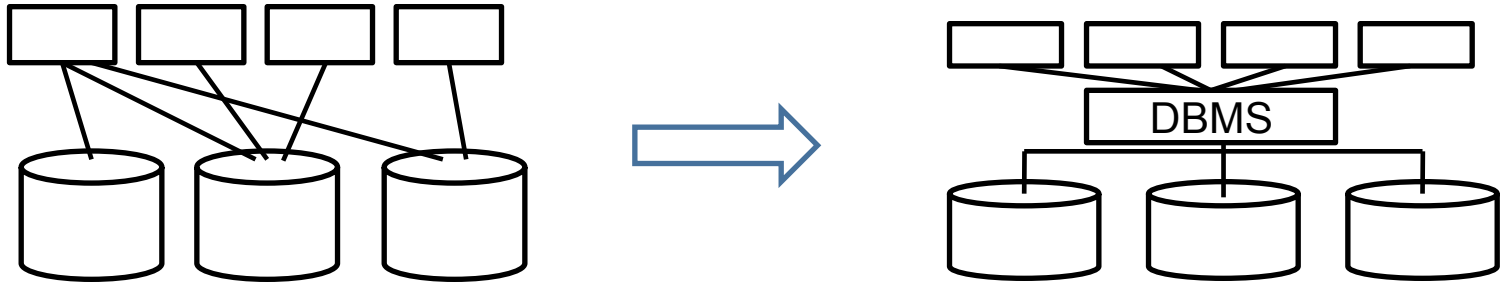
First came a program…

Once upon a time there was a program…



Direct access storage: late 60's

DBMS emerged as specialized programs for centralized data management



enova

Since then we have…

-Imperative programming languages, which tells, *how* to do things

```
for (i:=1, i++, n) do
…
end;
```

and

- Declarative data manipulating languages, which define *what* to do:

```
SELECT  first_name, last_name FROM people
WHERE id=101
```

So – is anything wrong with that?...

Both imperative programming languages and declarative query languages work perfectly to accomplish the tasks the were designed to accomplish.

The problems start, when we try to make them to work *together*.

enova

What we have at Enova:

Postgres: - RDBMS

Ruby: object-oriented language.

ActiveRecord - Object Relational Mapping (ORM)

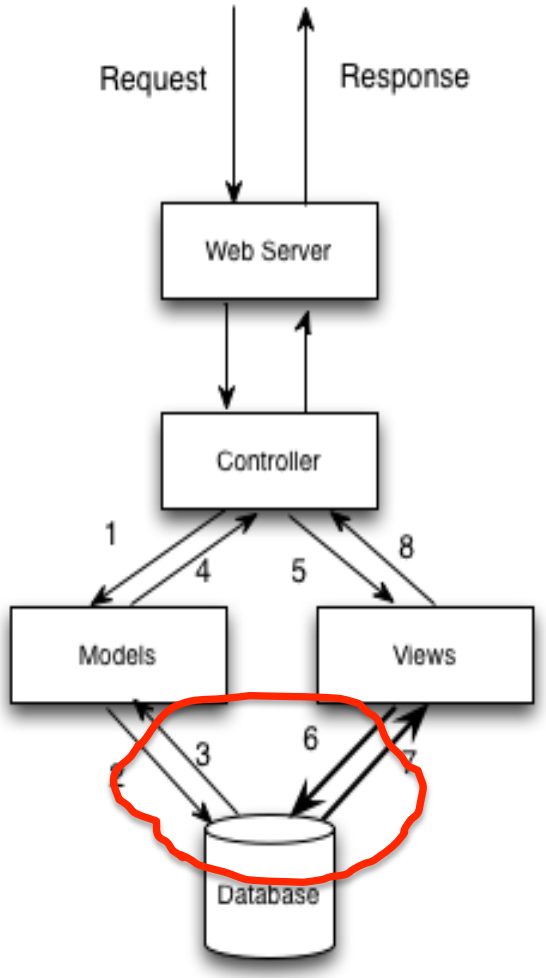
ORM:

- Data structures mapping – yes
- Data sets manipulation – no

ORIM – object-relational impedance mismatch

enova

How ActiveRecord works



What this means for application/database interaction

Due to the lack of awareness of the underlying database interaction on the part of the object methods, one controller performs multiple trips to the database

For example…

enova

A customer comes to the website…

After (s)he logs in…

Account History			
Line of Credit #	**Opened Date**	**Line of Credit Limit**	**Status**
35618598	08/18/2016	$1000.00	Paid Off
36220581	01/08/2014	$1650.00	Paid Off
35792230	10/30/2013	$1000.00	Declined
35726218	10/21/2013	$1000.00	Paid Off
35323358	08/16/2013	$800.00	Paid Off

Loan History				
Loan Type	**Loan ID**	**Funding Date**	**Loan Amount**	**Status**
Payday Loan	34289957	01/30/2013	$250.00	Paid Off
CAB Loan	32826262	06/04/2012	$700.00	Paid Off
CAB Loan	32792390	05/29/2012	$700.00	Paid Off
CAB Loan	32672252	05/07/2012	$700.00	Paid Off
Installment Loan	32254565	02/13/2012	$300.00	Paid Off

Manage My Account

- » Account Home
- » **Loan History**
- » Apply for a Loan
- » Announcements

Account Details

- » Personal Information
- » Bank Account
- » Employment
- » Contact Preferences
  - » New - Text Alerts!
- » Change Password

VeriSign Secured

Live Chat
Talk to us Now!

Resource Center

Refer a friend

enova

Account Presenter

```
def initialize(customer)
    @customer = customer
    @customer_extra = customer.customer_extra
    @person = customer.person
    @address = customer.person.try(:address)
    @company = customer.person.try(:company)
    @bank_account = customer.bank_account(true)
    @debit_card = customer.debit_card
    @customer_paydate = customer.customer_paydate(true)
    @paydate_schedule =
customer.customer_paydate.try(:paydate_schedule)
    @customer_source = customer.customer_source
  end
```

enova

Corresponding application log

```
SELECT * FROM customers
      WHERE (customers.id = 12470535)

SELECT * FROM people
    WHERE (people.id= 61657007 AND   (type =
'CustomerPerson')) AND ( (people.type =
'CustomerPerson' ) );

SELECT addresses.*, people_addresses.serial_number
FROM addresses
        INNER JOIN people_addresses ON addresses.id =
   people_addresses.address_id
 WHERE (people_addresses.person_id = 61657007
                AND (eff_end_date is NULL));
```

But wait, there's more!

```
SELECT * FROM approvals WHERE (customer_id = 12470535) ORDER BY processed_on desc
LIMIT 1
SELECT * FROM customers WHERE (customers.id = 12470535)
SELECT * FROM loans WHERE (loans.id = 25563928)
SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
('applied','approved','on_hold')) ORDER BY funding_date DESC LIMIT 1
SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
('applied','approved',E
'on_hold')) ORDER BY funding_date DESC LIMIT 1
SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
(‘issued','issued_pmt_proc')) ORDER BY funding_date DESC LIMIT 1
 SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
('applied','approved',E'on_hold')) ORDER BY funding_date DESC LIMIT 1
SELECT * FROM customers WHERE (customers.person_id = 61657007)
SELECT count(*) AS count_all FROM loans WHERE (loans.customer_id = 12470535 AND
(status_cd in ('applied','approved','on_hold','issued','issued_pmt_proc') and
loan_type_cd = 'installment'))
SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
('applied','approved','on_hold')) ORDER BY funding_date DESC LIMIT 1
SELECT * FROM loans WHERE (customer_id = 12470535 and status_cd in
('applied','approved','on_hold')) ORDER BY funding_date DESC LIMIT 1
```

enova

What we should see instead…

```
SELECT *
FROM customers
WHERE (customers."id" =
12470535)

SELECT *
FROM people
WHERE (people."id" =
61657007
AND (type =
'CustomerPerson'))
AND ( (people."type"
= 'CustomerPerson' ) );
```

```
SELECT addresses.*,
people_addresses.serial_number
FROM addresses
INNER JOIN people_addresses
ON addresses.id =
people_addresses.address_id
WHERE
(people_addresses.person_id =
61657007
AND (eff_end_date is NULL));
```

enova

There *are* some ways to improve

Eager loading:

```
SELECT customers."id"
      ,customers."created_by"
      <..>,customers."person_id"
      <..>,customers_extra."estimated_monthly_living_expense"
      <..>,people."type"
      <...>,addresses."id"<...>
      ,companies."id" <...>
FROM customers
    LEFT OUTER JOIN customers_extra
      ON customers_extra.customer_id = customers.id
     LEFT OUTER JOIN people
      ON people.id = customers.person_id AND people."type" = 'CustomerPerson'
    LEFT OUTER JOIN people_addresses ON people_addresses.person_id =
    people.id LEFT OUTER JOIN addresses ON addresses.id = <…>
WHERE (customers."id" = 17674188)
```

enova

But then the next method

```
has_many :bank_accounts, :foreign_key => 'person_id' do
    def default_for_customer(customer, reload = false)
      @bank_account_cache = {} if @bank_account_cache.nil? || reload
      <…>
  end
```

executes SELECT again!

If we continue with existing frameworks…

… this problem (ORIM) will never be solved, and we will continue to loose money on timeouts!

enova™

Our Solution: Logic Split Unfolded

enova

Our approach allows:

- ✓reduce the number of db calls

  (2-10 instead of 500-900 per view rendering)
- ✓optimize queries independently from the app.

enova

How we are going to achieve that?

Making the methods data-aware

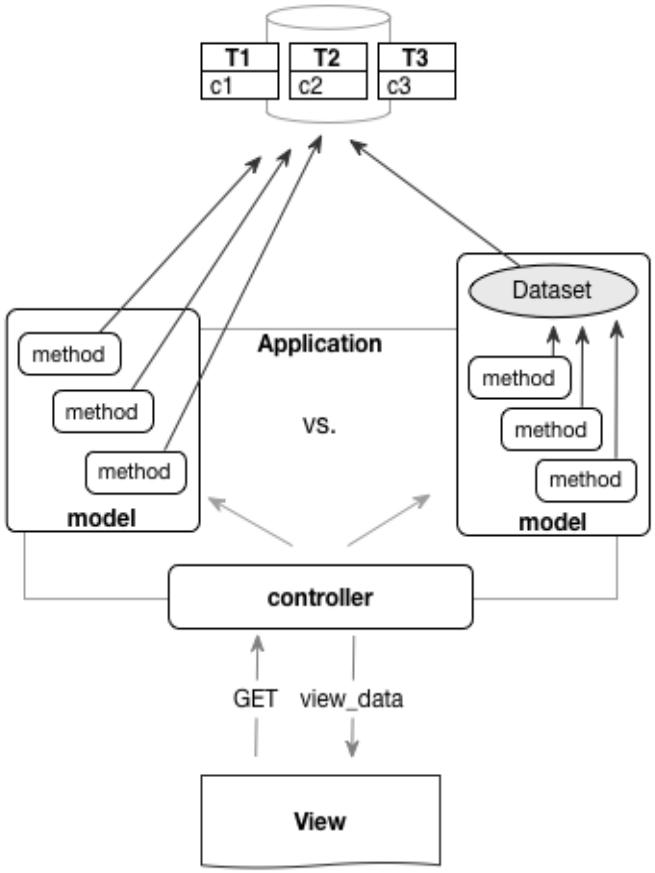
Contrary to the standard OO approach?

Yes, but…

This is the only way to improve the App/DB interaction.

enova

The sketch of proposed changes

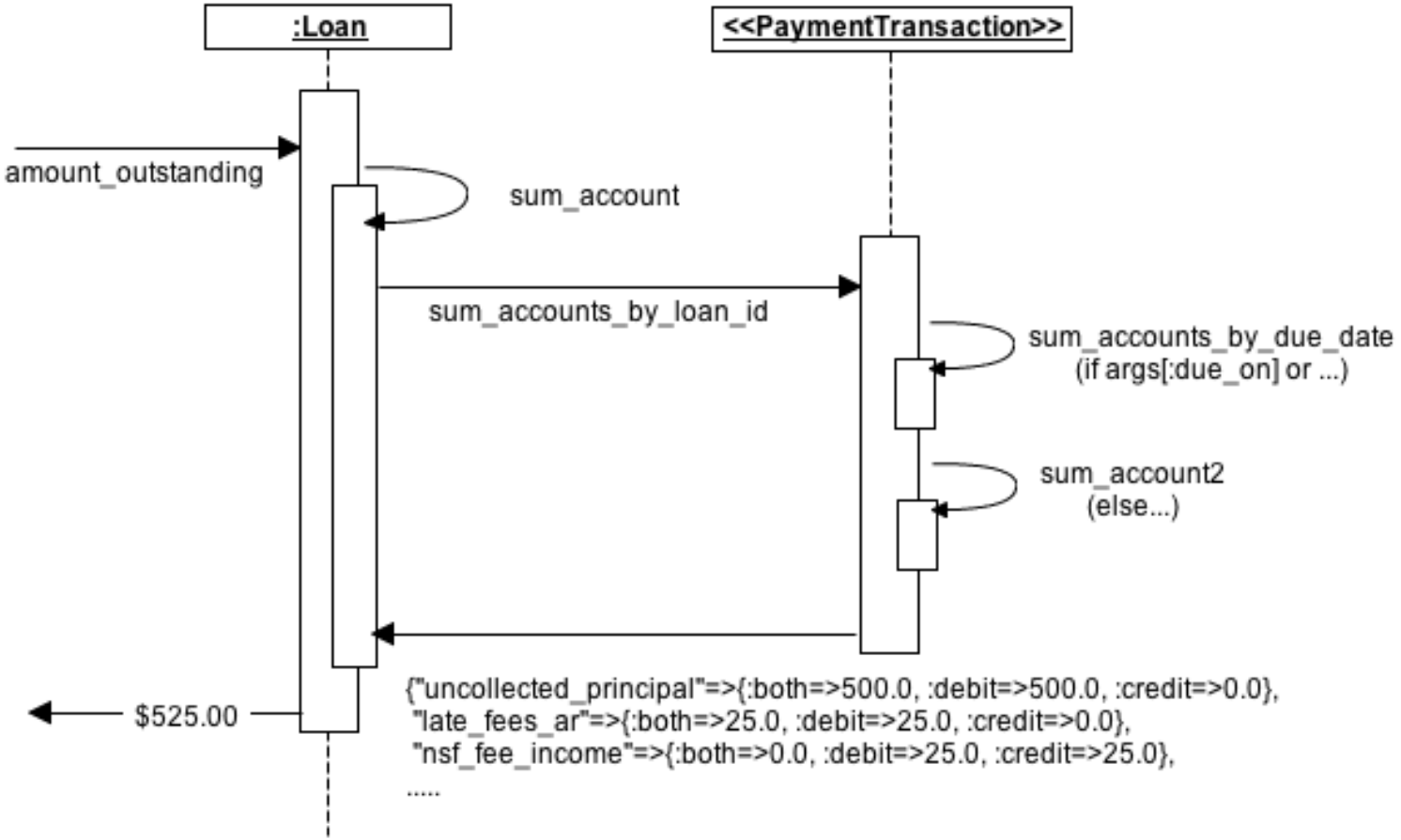


enova

Logic Split methodology

- ✓ Disassemble
- ✓ Identify data retrieval
- ✓ Construct a single query
- ✓ Execute
- ✓ Use retrieved data in other steps

Example: Amount_Outstanding



Definitions

AccountsOutstanding=

AccountsUncollected +

FeesOutstanding +

InterestOutstanding +

PrincipalAccounts +

AccountsDue

In turn:

- AccountsUncollected =

  uncollected_principal +

  uncollected_installment_principal

enova

Under the hood: database calls

```
SELECT
 vl.value AS account
,SUM(CASE vl.value WHEN pt.debit_account_cd
                   THEN pt.amount ELSE 0 END)
- SUM(CASE vl.value WHEN pt.credit_account_cd
                    THEN pt.amount ELSE 0 END) AS sum
FROM payment_transactions pt
JOIN valuelists vl ON vl.type_cd ='transaction_account'
 AND vl.value IN (pt.debit_account_cd,
pt.credit_account_cd)
AND loan_id=?
```

… and then a value for specific account is selected.

enova

Drawbacks

The method itself would allow retrieving all the information related to one loan “in one shot”.

However, because the application developers are unaware of the underlying layers, there *appears* to be no difference:

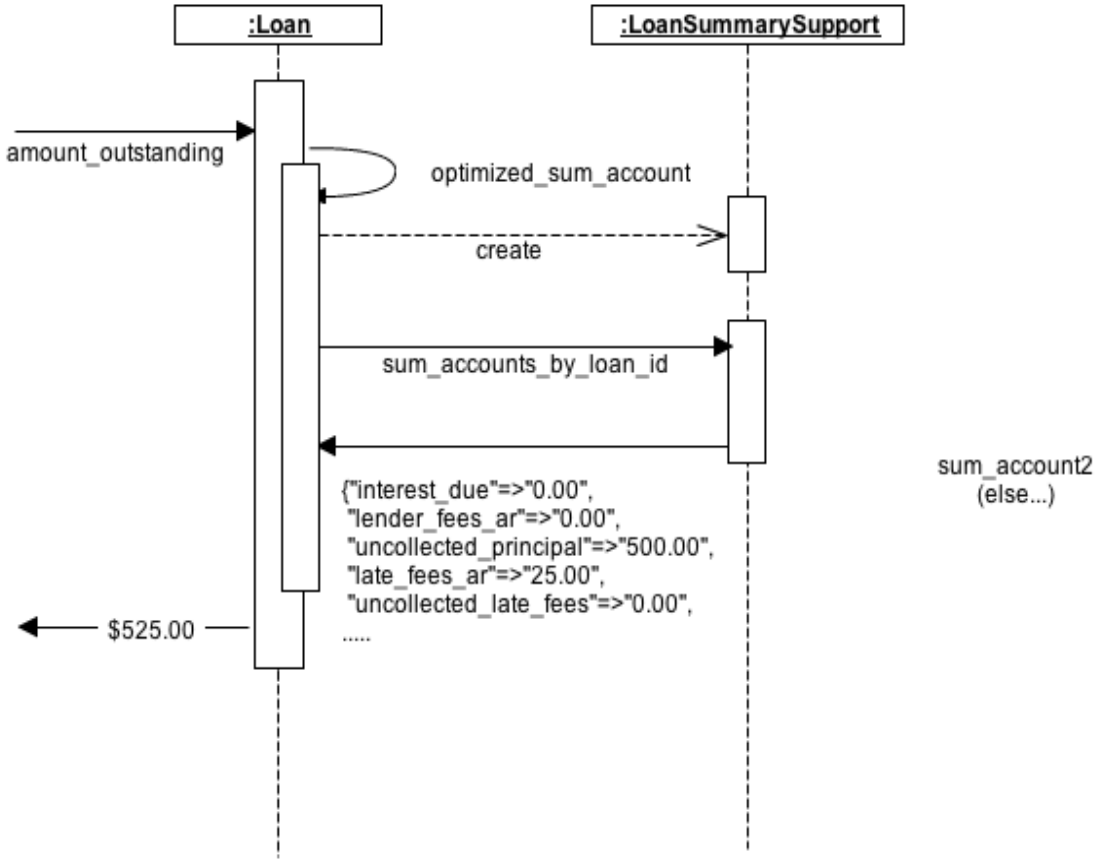
- Whether we obtain the values of all account balances one-by-one by following normal Object-Oriented method logic, in an imperative way
- Or if we obtain them all, simultaneously

Now for some **results**:

PGBadger log

2	**3h05m12s**	524,027	21.21	SELECT vl.value AS account, SUM(CASE vl.value WHEN pt.debit_account_cd THEN pt.amount ELSE 0 END) - SUM(CASE SUM(CASE vl.value WHEN pt.debit_account_cd THEN pt.amount ELSE 0 END) AS debit, SUM(CASE vl.value WHEN pt.credit_account_cd THEN pt.amount ELSE 0 END) AS credit FROM payment_transactions_committed pt INNER JOIN valuelists vl ON vl.type_cd = '' AND vl.value IN (pt.debit_account_cd, pt WHERE (loan_id = 0 AND installment_id = 0 AND committed = true) GROUP BY vl.v Show examples
3	**1h55m29s**	166,544	41.61	SELECT vl.value AS account, SUM(CASE vl.value WHEN pt.debit_account_cd THEN pt.amount ELSE 0 END) - SUM(CASE SUM(CASE vl.value WHEN pt.debit_account_cd THEN pt.amount ELSE 0 END) AS debit, SUM(CASE vl.value WHEN pt.credit_account_cd THEN pt.amount ELSE 0 END) AS credit FROM payment_transactions_committed pt INNER JOIN valuelists vl ON vl.type_cd = '' AND vl.value IN (pt.debit_account_cd, pt WHERE (loan_id = 0 AND committed = true) GROUP BY vl.value; Show examples

Modified method



Under the hood: database calls

```
SELECT  loan_id
      , sum(CASE WHEN debit_account_cd = 'uncollected_principal'
                      THEN pt.amount ELSE 0  END
-CASE WHEN credit_account_cd='uncollected_principal'
                  THEN pt.amount ELSE 0  END) AS uncollected_principal
<...>
, sum(CASE WHEN debit_account_cd = 'uncollected_nsf_fees'
                 THEN pt.amount ELSE 0  END
- CASE WHEN credit_account_cd = 'uncollected_nsf_fees'
       THEN pt.amount ELSE 0  END) AS uncollected_nsf_fees
      , sum(CASE WHEN debit_account_cd = 'installment_principal'
                      THEN pt.amount ELSE 0  END
- CASE WHEN credit_account_cd = 'installment_principal'
     THEN pt.amount ELSE 0  END) AS installment_principal
  FROM payment_transactions_committed pt
             INNER JOIN loans l ON l.id=pt.loan_id
WHERE loan_id={?}
```

Execution statistics from dark testing: old

4.895s	3,350	0.001s/0.005s/0.001s	`SELECT vl.VALUE AS account, sum (CASE vl.VALUE WHEN pt.debit_account_cd THEN pt vl.VALUE WHEN pt.credit_account_cd THEN pt.amount ELSE 0 END) AS sum, sum (CAS pt.debit_account_cd THEN pt.amount ELSE 0 END) AS debit, sum (CASE vl.VALUE WH pt.amount ELSE 0 END) AS credit FROM payment_transactions pt JOIN valuelists vl IN (pt.debit_account_cd, pt.credit_account_cd) WHERE (loan_id = 0 AND install` Show examples
4.625s	844	0.000s/0.259s/0.005s	`UPDATE genesys.channel_messages SET "message_identifier" = e '', "request_messag "channel" = e '', "owner_id" = 0, "response_message" = e '', "response_time" = e` Show examples

… and new

1.861s	985	0.001s/0.006s/0.002s	`SELECT * FROM loans.sum_accounts_by_loan_id_detailed ( NULL, 0, NULL, NULL, 0, NULL, );` Show examples
1.822s	515	0.001s/0.316s/0.004s	`SELECT rv.* FROM lexis_nexis.risk_view_reports rv INNER JOIN credit_reports cr ON cr. cr.report_type_cd = '' WHERE ( cr.customer_id = 0 ) ORDER BY cr.inquiry_time DESC LIM` Show examples
1.758s	144	0.003s/0.039s/0.012s	`SELECT sum ( heap_blks_read ) AS a1, sum ( heap_blks_hit ) AS a2, sum ( idx_blks_read ) AS a4, sum ( toast_blks_read ) AS a5, sum ( toast_blks_hit ) AS a6, sum ( tidx_blks tidx_blks_hit ) AS a8 FROM pg_statio_user_tables;` Show examples

So… now everybody happy?...

…not really…

What do application developers say at this point?....

Wait! What about the business logic?!

enova

What about the business logic?

- ✓We need some business logic to execute joins and selects
- ✓Selected results transformations and manipulations do not have to be executed on the database side.

enova

One more time:

- ✓ Disassemble method into atomic steps,
- ✓ Identify ones which require data retrieval
- ✓ Using knowledge about database objects relationships, construct a single query
- ✓ Execute
- ✓ Use retrieved data in other steps

Let’s review another example – account balance calculation for Lines of credit.

enova

Account_Balance method for LOC

- Obtain account principal balance
- Obtain outstanding fees and interest as of next payment due date
- Calculate the interest credit (unearned interest) for the number of days left before the payment due date
- Obtain existing customer balance
- Calculate the total account balance using the values obtained on steps 1-4.

enova

Traditional object-oriented approach

Account_Balance method calls:

- Principal_Balance,
- Interest_Amount,
- Fees_Amount,
- Customer_Balance
- Interest_Credit

Each of them would interact with a database **independently.**

enova

Drilling down into each of the steps

1. Principal balance as described for account_outstanding requires a single database call.
2. Outstanding interest and fees require one database call each.
3. Interest credit calculation:
   - 3.1. Obtain the daily interest rate for this customer
   - 3.2. Obtain base amount, which is used to calculate the total interest
   - 3.3. Obtain the number of days, for which the interest should be credited:
     - 3.3.1. Obtain the next payment due date
     - 3.3.2. Calculate number of days based on obtained date and todays' date
   - 3.4. Calculate amount of credit, based on results from the previous three steps
4. Customer balance can be obtained using one database call, same as steps 1-3.

enova

Combining steps with data retrieval

For a given loan, retrieve payment transactions, which show principal balance, current interest, fees and customer balance, also retrieve loan's daily interest rate and next payment due date.

After disassembling this method..

… we were able to retrieve all data using one single SELECT statement and two simple Ruby methods.

SELECT statement

```
SELECT l.id AS loan_id
        ,sum( CASE WHEN debit_account_cd = ‘principal’
         AND t.acct_date<= v_current_date
         THEN t.amount  ELSE 0  END
                  - CASE WHEN credit_account_cd
=’principal’
     AND t.acct_date<= v_current_date
    THEN t.amount ELSE 0  END  )  AS amount_payable
       ,sum (CASE WHEN
t.debit_account_cd=’fees_provisional’
                        THEN t.amount ELSE 0 END
- CASE WHEN t.credit_account_cd= ‘fees_provisional’
            THEN t.amount ELSE 0 END )
 AS fees_provisional
       ,sum (CASE WHEN t.debit_account_cd=’
interest_provisional’
                         THEN t.amount ELSE 0 END
- CASE WHEN t.credit_account_cd =’interest_provisional’
            THEN t.amount ELSE 0 END )
AS interest_provisional
      ,st.end_date AS next_closing_date
     ,l.daily_rate AS interest_rate
     ,sum (CASE WHEN t.debit_account_cd
='customer_balance'
                        THEN -t.amount ELSE 0 END
 - CASE WHEN t.credit_account_cd ='customer_balance'
          THEN -t.amount ELSE 0 END )
    AS customer_balance
FROM loans l
    LEFT OUTER JOIN payment_transactions_committed
t ON
                        l.id=t.loan_id
   LEFT OUTER JOIN statements st ON
                        l.id=st.loan_id
      WHERE l.id={?}
     GROUP BY l.id
                ,l.daily_rate
                ,st.end_date
```

enova

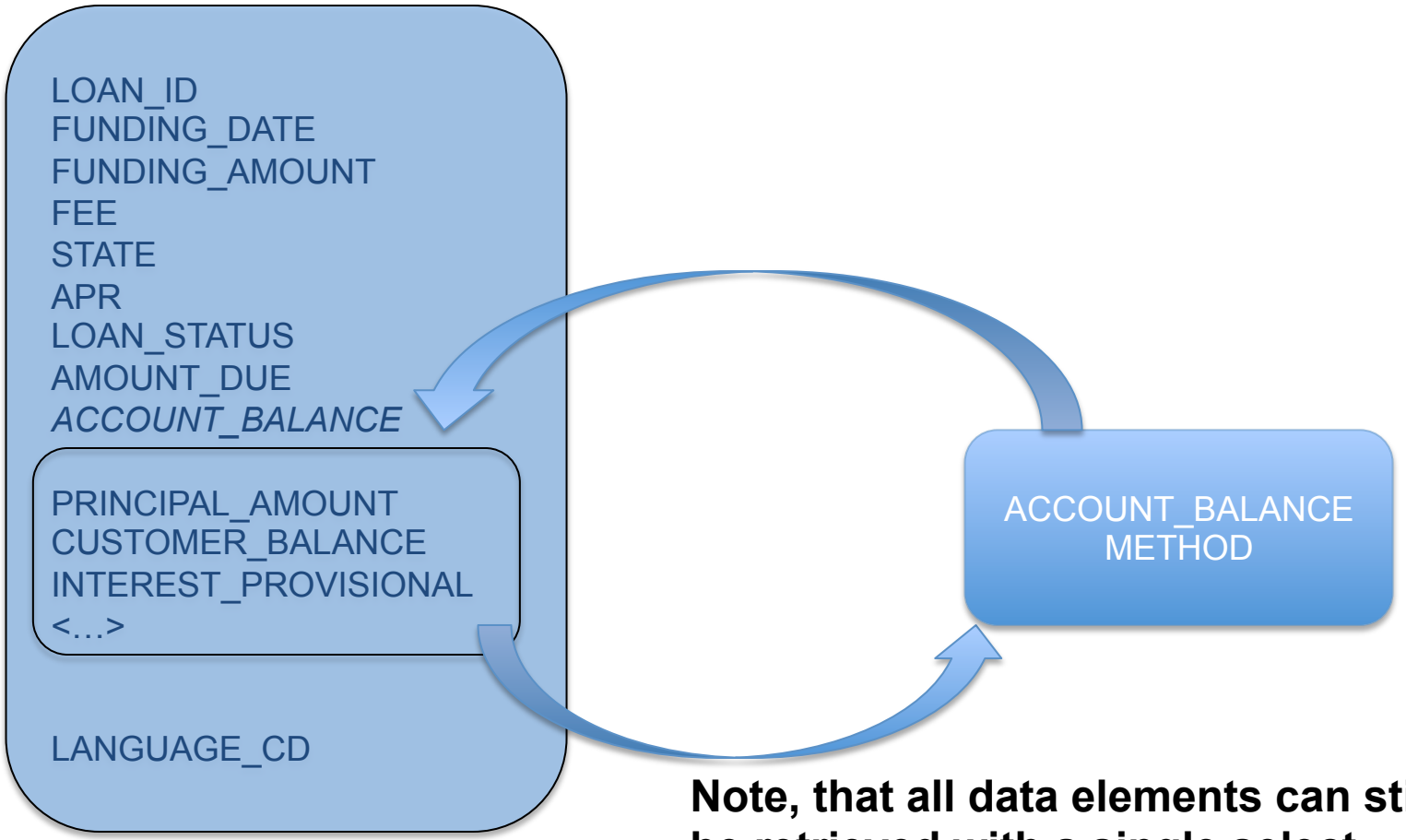
Ruby Method: account_balance

```
def account_balance
    <..>
  amt = amt + provisional_fees +
[ provisional_interest - [unearned_interest,
0].max, 0 ].max
    <…>
    return [0, amt].max
  end
```

enova

Ruby Method: unearned_interest

```
def unearned_interest
    amt = -1 * (principal_amount +
[(customer_balance - interest_provisional -
fees_provisional), 0].max)
    next_closing_date =
Date.parse(self.next_closing_date)
    <..>
    interest < 0.0 ? 0 : interest
  end
```



Note, that all data elements can still be retrieved with a single select statement, which won't be possible within the standard ORM framework.

On a Larger Scale

enova

How satisfied the users are…

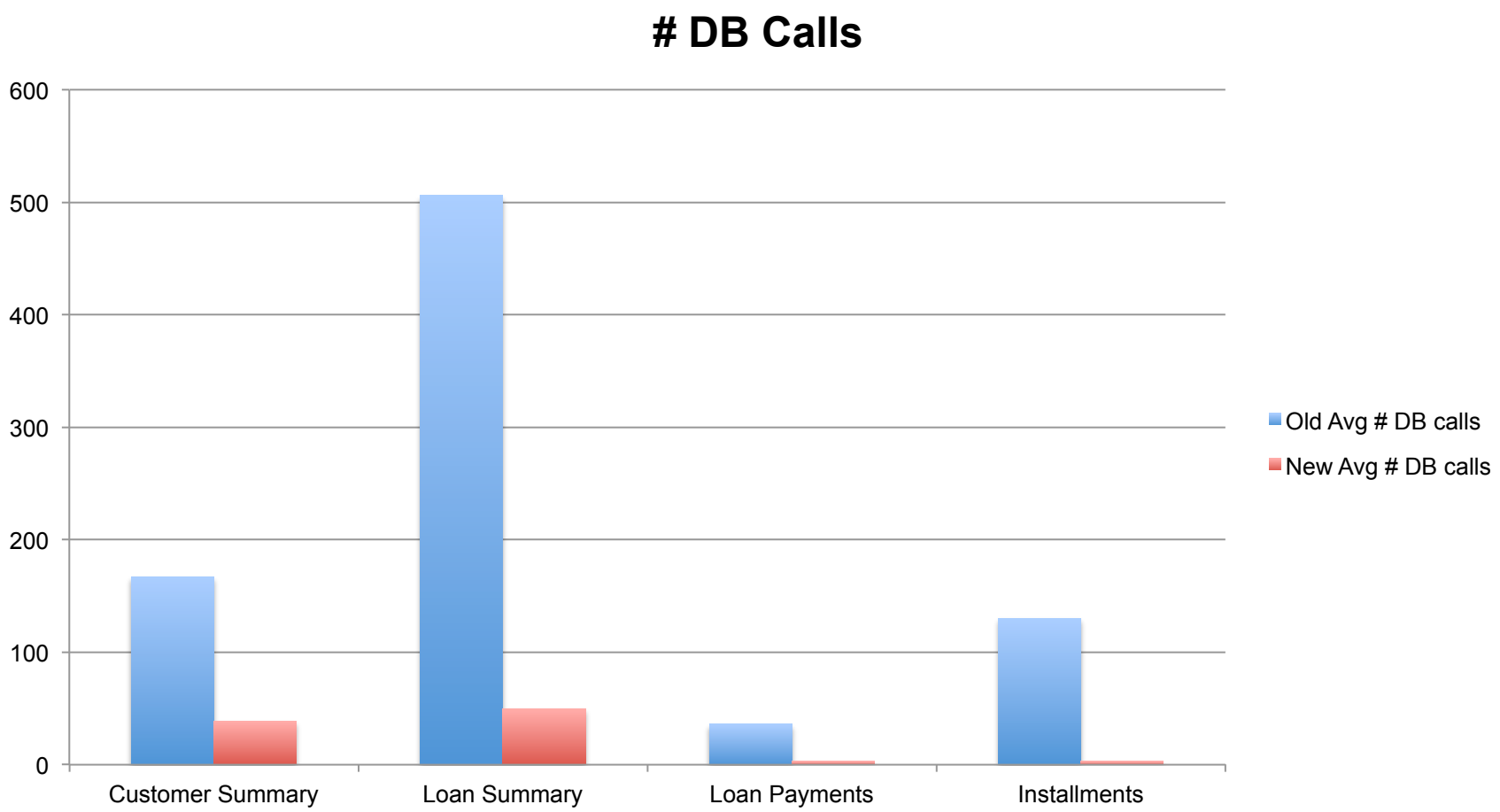
Numbers from our weekly report:

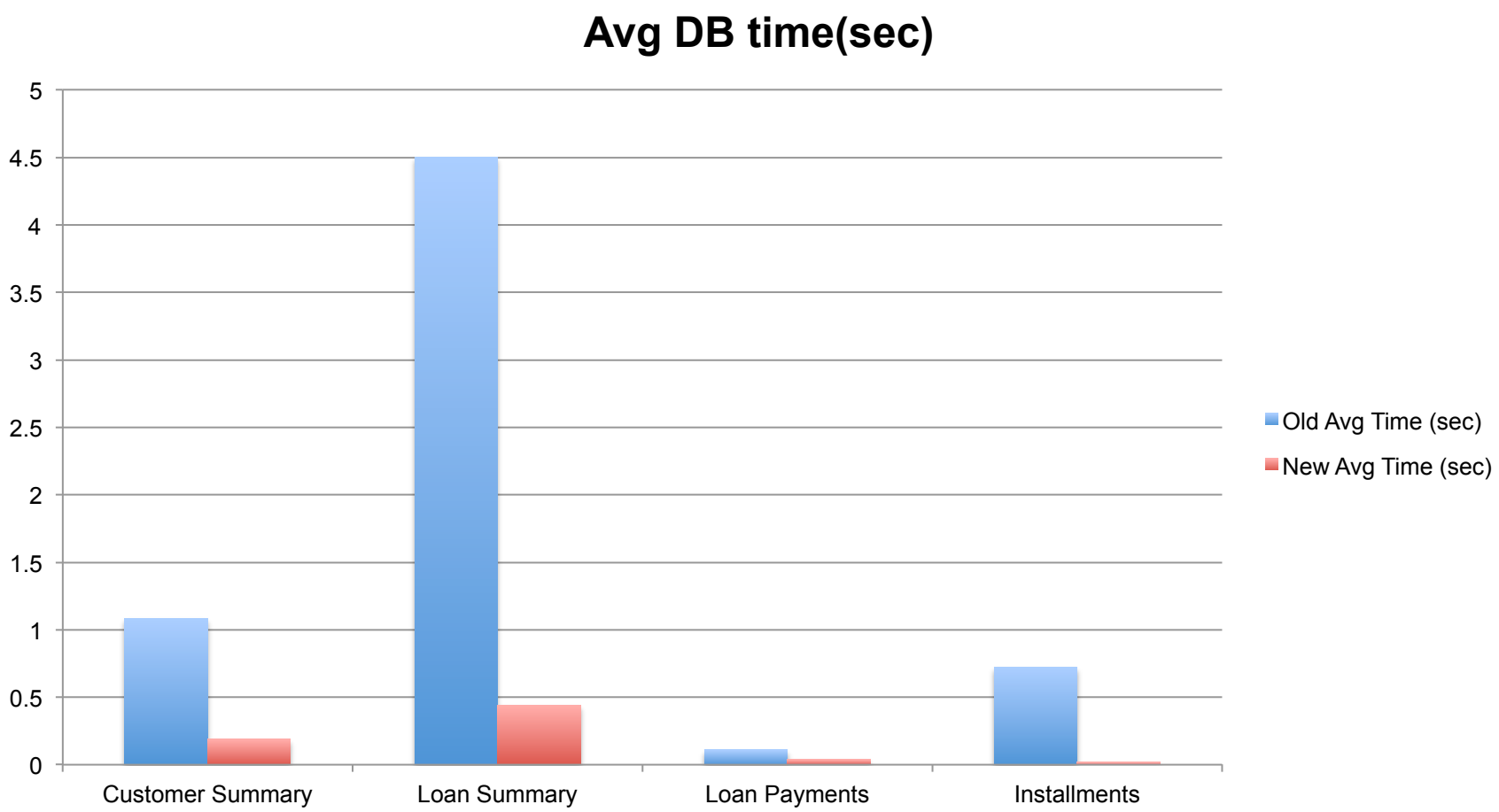
APP	Page Views (thousands)	Load Time (sec)	% Satisfied
Old APP 1	21.7	6.83	64.2
Old APP 2	29.5	5.29	64.4
New APP	26.5	0.514	99.8

enova

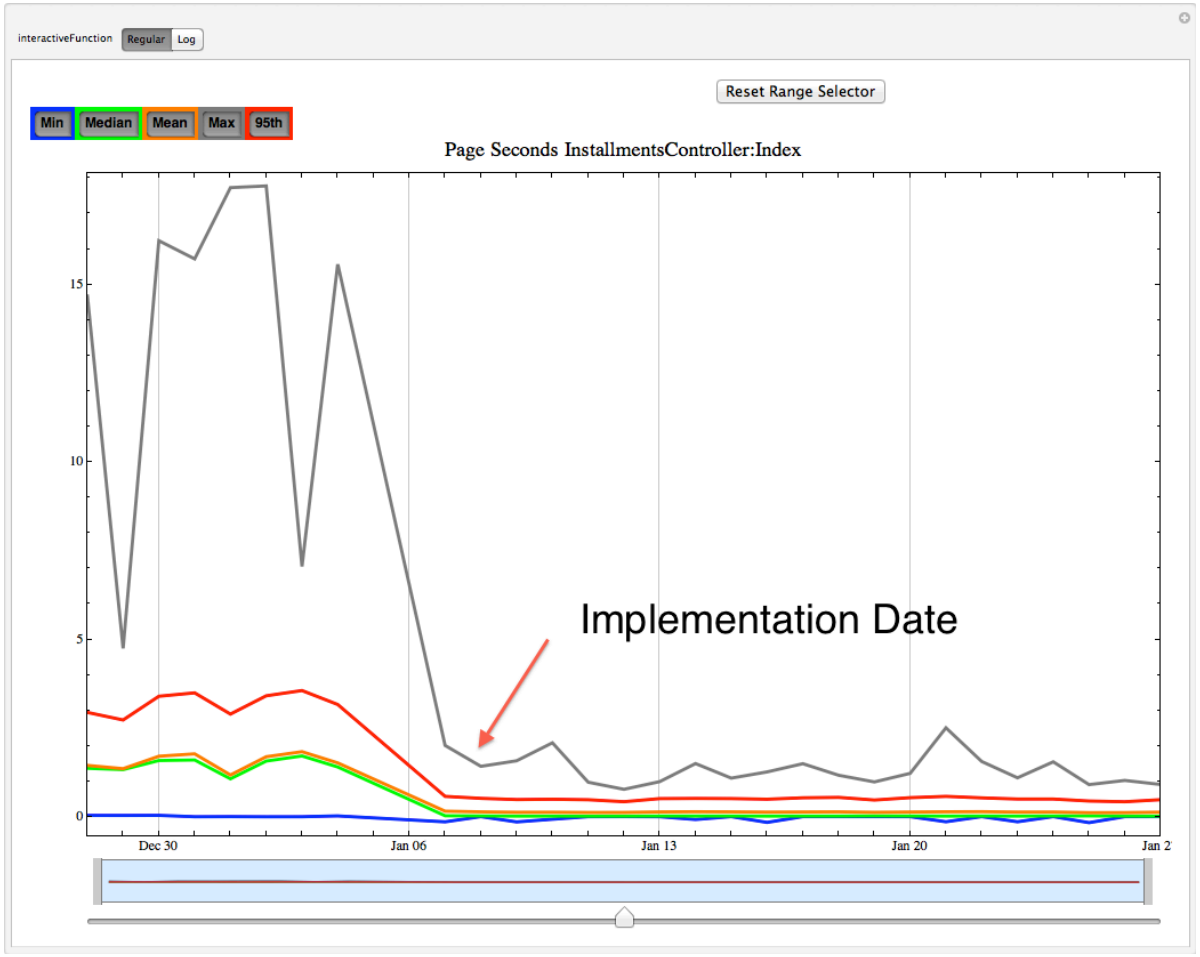
Execution statistics: Old App vs. New App

Controller action	Old Avg # DB calls	New Avg # DB calls	Old Avg Time (sec)	New Avg Time (sec)
Customer Summary	167	39	1.08	0.19
Loan Summary	506	50	4.5	0.44
Loan Payments	36	3	0.11	0.04
Installments	130	3	0.72	0.018


DB Calls
600
500
400
300
200
100
0
Customer Summary
Loan Summary
Loan Payments
Installments
Old Avg # DB calls
New Avg # DB calls


Avg DB time(sec)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Customer Summary
Loan Summary
Loan Payments
Installments
Old Avg Time (sec)
New Avg Time (sec)

Installment Presenter page load time



Now

- We save both time
- *And* money!

Related Work

Were we the first to notice the problem?

Definitely ***NOT!***

The problem of object-relational impedance mismatch is a constant discussion topic.

In recent years multiple attempts were made to try to resolve this issue with no significant outcome.

enova

Hibernate ORM Service

- Claim: does not hide “the power of SQL” from developers.
- Yes, it *allows* queries

But…

- It is not an easy task
- Still prompts “natural” solutions

enova

Newer Version of Active Record

- Allows Eager loading and some customer queries, but has the same limitations as Hibernate

- Eager loading may cause an excessive application memory usage

enova

Agile Technology

- acknowledges the existence of ORIM ;
- not only the technical impedance mismatch, but also a cultural impedance mismatch:

  > "The object-oriented paradigm is based on proven software engineering principles. The relational paradigm, however, is based on proven mathematical principles".

- raises awareness of the problem

But…

Agile data technology solutions:

- schema changes and/or more careful design – *yes*
- data refactoring – *yes*
- dealing with inefficient queries - *no*

enova

Other related work

- AppSlueth tool for application tuning: identifies "delinquent design patterns"; in general does not allow to stay within ORM, or to reuse existing methods
- SQLAlchemy tool: allows the object model and database schema to develop in a cleanly decoupled way from the beginning; problem: implies that an application developer is at the same time a database developer, is aware of the best data access paths, and can divert from the OO design standards
- Dbridge project: holistic approach to the application optimization.

What's Next?

enova

Future work

Our goal: to make the Logic Split a company-wide development methodology

Problems:

- Large amount of the legacy code
- No business specifications
- Evolving legacy app
- The human factor

enova

Tasks:

- ✓Separating logic from the Ruby code
- ✓Verifying it with business stakeholders;
- ✓Reusing of our new models
- ✓Continue rewriting
- ✓Clarifying technology
- ✓Constant results measuring

And let me give you just one example…

UK portal current execution times - Loan Controller:

Loan type	Avg DB Calls	Max DB Calls	Avg. Page Time	Avg. DB Time	Max DB Time
payday	388	5292	2.83	2.06	40.37
installment	1112	10863	10.32	6.74	73.53
oec	274	1594	2.65	1.76	12.50

loan id=6820002, db time: 73.53 sec

SELECT * FROM aperture.loan_search_single_loan (6820002) – 0.26 sec
SELECT * FROM aperture.loan_installments__api (6820002) – 0.03 sec
SELECT * FROM aperture.loan_payments__api (6820002) – 0.2 sec

enova

Acknowledgements

Many thanks to:

- Enova CIO/CTO Fred Lee
- Team Raven, and everybody who worked on the Aperture project during it's lifetime:
  - Jef Jonjevic
  - Marc Groulx
  - Richard Lee
  - Preeti Dhiman
  - Luda Baklanova
  - Jian Wu
  - Chetana Yogeesh
  - Alan Zoppa
  - Ana Lebon
  - Kevin Glowacz
  - Neha Bhardwaj
  - Ranning Li
- Sheldon Strauch and the entire DB Dev team
- Chad Slaughter and Kurt Stephens
- Ben Heilman, Donny Jekels and Sam Rees
- Dave Trollop and Paul Solomon

Find more at:

- www.enova.com
- The World of Data: hdombrovskaya.wordress.com
- hdombrovskaya@enova.com