POSTGRES MVCC

A DEVELOPER CENTRIC VIEW OF MULTI VERSION CONCURRENCY CONTROL

By: Robert Sosinski

Reactive.IO

Robert Sosinski

Founder & Engineering Fellow

AGENDA

MVCC: what it is and it matters

Transactions: more then just an undo button

Isolation Levels: seeing what you need to see

Locking: control when your data is written

Cursors: stream chronologically correct data

Summary: bringing it all together

Questions: ready, fire, aim

WHAT IS MVCC?

Multi

Version

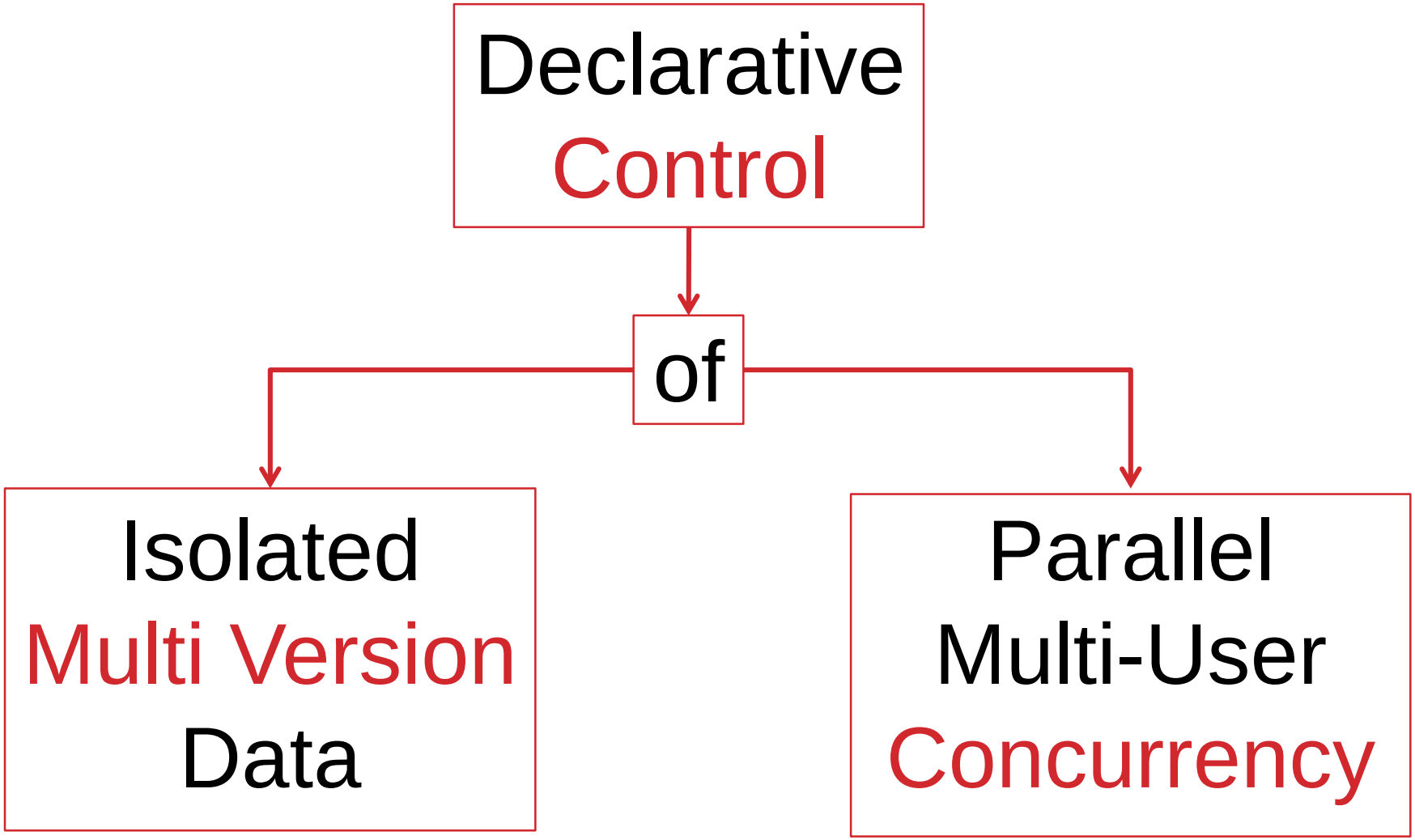
Concurrency

Control

WHAT IS MVCC?

Multi **V**ersion
Concurrency
Control

WHAT IS MVCC USING BOXES AND ARROWS



WHAT DOES MVCC DO?

1: Multiple users are able access the same data at the same time.

2: Every user sees their own isolated snapshot of the database.

3: Changes made by one user, will not be seen by any other user until their transaction is committed.

THE MVCC SALES PITCH

MULTI VERSION	CONCURRENCY
Atomic Updates	Higher efficiency
Consistent Data	Simpler operations
Isolated Reads	Engineering agility

A MVCC WORLD


MySQL


H2


Microsoft
SQL Server

ORACLE


MarkLogic


Lucene


Apache
Solr

A WORLD WITHOUT MVCC

Simple HR database

for a fictitious company

with high employee churn

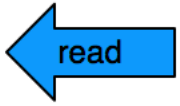
without MVCC.

UPDATE IN PLACE 1: TABLE

id	name	notes
1	Alice	Great at programming
2	Bob	Always talking to alice
3	Eve	Listens to everyone's conversations

UPDATE IN PLACE 2: SCAN

id	name	notes
1	Alice	Great at programming
2	Bob	Always talking to alice
3	Eve	Listens to everyone's conversations



UPDATE IN PLACE 3: UPDATE

	id	name	notes
	1	Alice	Great at programming
~ update	2	Bob	Always talking to alice
	3	Eve	Listens to everyone's conversations

UPDATE IN PLACE 4: UPDATED

	id	name	notes
	1	Alice	Great at programming
~ update	2	Bob	**Working very hard**
	3	Eve	Listens to everyone's conversations

UPDATE IN PLACE 5: INSERT

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
3	Eve	Listens to everyone's conversations
4	**Dave**	**Very promising new-hire**

+ insert

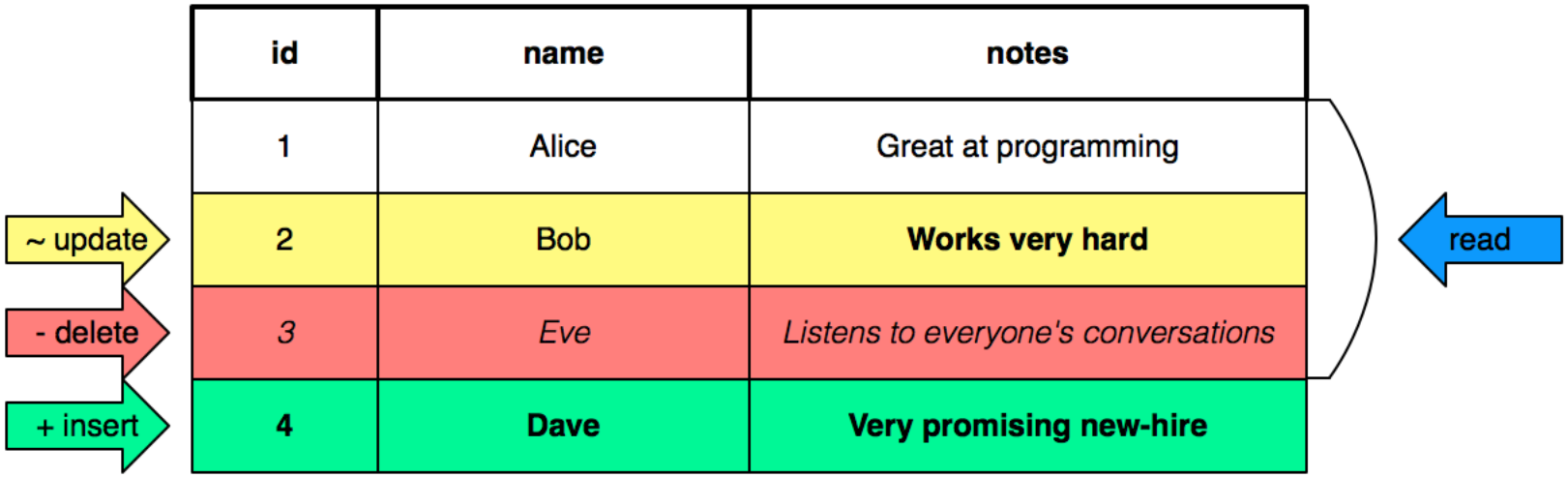
UPDATE IN PLACE 6: DELETE

	id	name	notes
	1	Alice	Great at programming
	2	Bob	**Working very hard**
- delete	3	Eve	Listens to everyone's conversations
	4	**Dave**	**Very promising new-hire**

UPDATE IN PLACE: 7

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
4	**Dave**	**Very promising new-hire**

UPDATE IN PLACE 8: REALITY



HOW TO SOLVE THIS PROBLEM

Pessimistic locking: **lock everything during writes**

Imperative controls: **synchronization and mutexes**

System build out: **everyone gets their own database**

Let the cards fall: **whatever happens, happens…**

HOW TO SOLVE THIS PROBLEM

Pessimistic locking: **lock everything during writes**

Imperative controls: **synchronization and mutexes**

System build out: **everyone gets their own database**

Let the cards fall: **whatever happens, happens**

MVCC: **Let the database handle the particulars**

MVCC 1: TABLE

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	0	2	Bob	Always talking to alice
102	0	3	Eve	Listens to everyone's conversations

TXID
103

MVCC 2: UPDATE

	xmin	xmax	id	name	notes
	100	0	1	Alice	Great at programming
- update	101	0	2	Bob	Always talking to alice
	102	0	3	Eve	Listens to everyone's conversations

TXID
103

MVCC 3: UPDATE IN PROGRESS

	xmin	xmax	id	name	notes
	100	0	1	Alice	Great at programming
- update	*101*	***103***	*2*	*Bob*	*Always talking to alice*
	102	0	3	Eve	Listens to everyone's conversations

TXID
103

MVCC 4: UPDATE IN PROGRESS

	xmin	xmax	id	name	notes
	100	0	1	Alice	Great at programming
	101	***103***	*2*	*Bob*	*Always talking to alice*
	102	0	3	Eve	Listens to everyone's conversations
+ update	**103**	**0**	**2**	**Bob**	**Working very hard**

TXID
103

MVCC 5: UPDATED

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**

TXID
104

MVCC 6: INSERT

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**
+ insert 104	0	**4**	**Dave**	**Very promising new-hire**

TXID
104

MVCC 7: INSERTED

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	0	3	Eve	Listens to everyone's conversations
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

TXID
105

MVCC 8: DELETE

- delete

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	*103*	*2*	*Bob*	*Always talking to alice*
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

TXID
105

MVCC 9: DELETED

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

TXID
106

LETS CONCURRENTLY WRITE SOME DATA

Open Postgres Terminal

VACUUM 1: UIP TABLE

id	name	notes
1	Alice	Great at programming
2	Bob	**Working very hard**
4	**Dave**	**Very promising new-hire**

VACUUM 2: MVCC TABLE

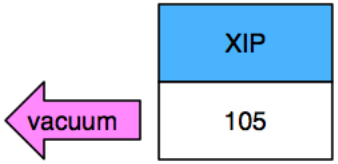
xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

VACUUM 3: WASTE

	xmin	xmax	id	name	notes
	100	0	1	Alice	Great at programming
:(waste	*101*	***103***	*2*	*Bob*	*Always talking to alice*
:(waste	*102*	***105***	*3*	*Eve*	*Listens to everyone's conversations*
	103	**0**	**2**	**Bob**	**Working very hard**
	104	0	**4**	**Dave**	**Very promising new-hire**

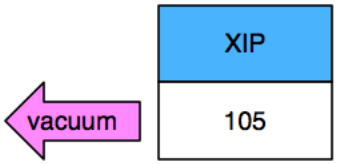
VACUUM 4: FIRST VACUUM

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
101	***103***	*2*	*Bob*	*Always talking to alice*
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**



VACUUM 5: FIRST VACUUM

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**



VACUUM 6: FIRST VACUUM

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

vacuum

XIP
105

VACUUM 7: VACUUM FINISHED

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

XIP
105

VACUUM 8: SECOND VACUUM

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

XIP

VACUUM 9: SECOND VACUUM

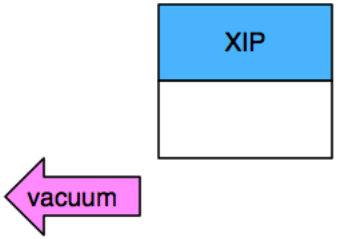
xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
102	***105***	*3*	*Eve*	*Listens to everyone's conversations*
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

vacuum

XIP

VACUUM 10: SECOND VACUUM

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**



XIP

VACUUM 11: VACUUM FINISHED

xmin	xmax	id	name	notes
100	0	1	Alice	Great at programming
103	**0**	**2**	**Bob**	**Working very hard**
104	0	**4**	**Dave**	**Very promising new-hire**

XIP

ISOLATION LEVELS

Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read Commited *Default*	✓			
Repeatable Read	✓	✓	✓	
Serializable	✓	✓	✓	✓

Referenced from: http://www.postgresql.org/docs/9.3/static/transaction-iso.html

✓ Not Possible

LETS ISOLATE SOME TRANSACTIONS

Open Postgres Terminal

EXPLICIT LOCKING

TABLE LEVEL	ROW LEVEL
Very broad	Very granular
8 types	2 types
Can affect querying	Will not affect querying

ROW LEVEL LOCKING

Name	Lock Type	Blocks Update	Blocks Select For Update
For Share	Row Share	✓	
select * from people where id = 1 for share;			
For Update	Row Exclusive	✓	✓
select * from people where id = 1 for update;			

LETS LOCK SOME ROWS

Open Postgres Terminal

CURSORS

Streaming: **break large datasets in smaller segments**

Efficient: **reduce a queries memory consumption**

Isolated: **return chronologically correct data**

Traversable: **can scan forward, backwards and more**

Flexible: **PL/pgSQL functions can return/accept cursors**

LETS USE A CURSOR

Open Postgres Terminal

SUMMARY

Powerful: interact with your data on your terms

Declarative: easy to use, less chance of mistakes

Efficient: use less resources to work with more data

Scalable: handle more processes with larger volume

Flexible: do what you need do when you need it

CONDENSED SUMMARY

Control of your Concurrent Multi Versioned Data

CONDENSED SUMMARY

Control of your Concurrent Multi Versioned Business

THANKS

Open For Questions