



Wvlet: A New Flow-Style Query Language

For Functional Data Modeling and Interactive Data Analysis

December 12, 2024
Trino Summit 2024

Taro L. Saito (Leo)
Senior Principal Engineer at Treasure Data

2024: 50th Anniversary of SQL

- SQL was born in 1974
 - SQL's early design choices paved the way for standardization and commercial adoption
 - Since then, applications of SQL have expanded significantly
 - OLAP: From RDBMS to Trino, Hive, Spark, etc.
 - Embedded DB: SQLite (OLTP) -> DuckDB (OLAP with Parquet/Iceberg support)
 - SQL's user base has grown beyond DBA to include non-engineers

research and advances

DOI:10.1145/2844827

A discussion of the evolution of the database industry over the past half century, and why the relational database concepts introduced by E.F. Codd have proven to be so resilient over several decades.

BY DONALD CHAMBERLIN

50 Years of Queries

E.F. CODD'S "A Relational Model of Data for Large Shared Data Banks"¹ is one of the most influential papers in all of computer science. In it, Codd defined concepts that are still in widespread use today, more than five decades later, including defining the theoretical foundation of the relational database industry.

When Codd's paper appeared in *Communications of the ACM* in June 1970, I was a student member of ACM, but I didn't receive the issue right away. I was driving cross-country from Stanford University to take a summer job at IBM's T.J. Watson Research Center in Yorktown Heights, New York. Before long, my summer job turned into a permanent IBM job, and I joined a group that was looking into the future of data management. My first task was to get up to speed on the current state of the art.

After being assigned to study Codd's paper in conjunction

The 1980 U.S. census. Punched cards had a successful 50-year product life, but they were largely replaced by magnetic tape. In the mid-20th century, data was typically stored on a magnetic tape and decoded sequentially by a tape drive. A tape might, for example, be used by an inventory-control application. Periodically, the data in the tape would be read sequentially, applying updates as it went along and producing a new, updated copy of the data. As a college student in 1964, I had a summer job as a computer operator, running jobs like this.

The advent of magnetic disks, introduced in 1956 by IBM RAMAC in 1956, changed everything. Now data was stored and processed. It was no longer necessary for applications to process data sequentially, since data on magnetic disks could be accessed directly in any order. This gave rise to a new wave of innovation in how data should be organized on disk.

In the 1960s, I worked on a project working on a NASA contract developing a disk-based information storage and retrieval system for use in the Apollo moon landing program.

Key insights

- The relational data model, proposed by E.F. Codd in 1970, is the most widely used format for business data.
- The relational model was first demonstrated in the 1970s by experimental prototypes developed at the University of California. The 1980s saw a proliferation of relational databases.
- SQL (short for SEQUEL) was designed in 1974 as a language for updating tables, but it has been used for many other purposes since its acceptance as an ISO standard in 1986. Availability of high-quality open-source relational databases has made SQL the most widely used query language.
- Current requirements for massive amounts of data are forcing us to move away from the traditional relational model.

International Standard
ISO/IEC 9073-1
Information technology - Database
Edition 2.0 (2012)
ISO/IEC JTC1/SC32

1973 Award
Oracle

The Programmer
Charles H. Bachman

A Relational Model of Data for Large Shared Data Banks

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

Future uses of large data banks must be planned to know how to organize the data in the most effective way. A prompt:

```

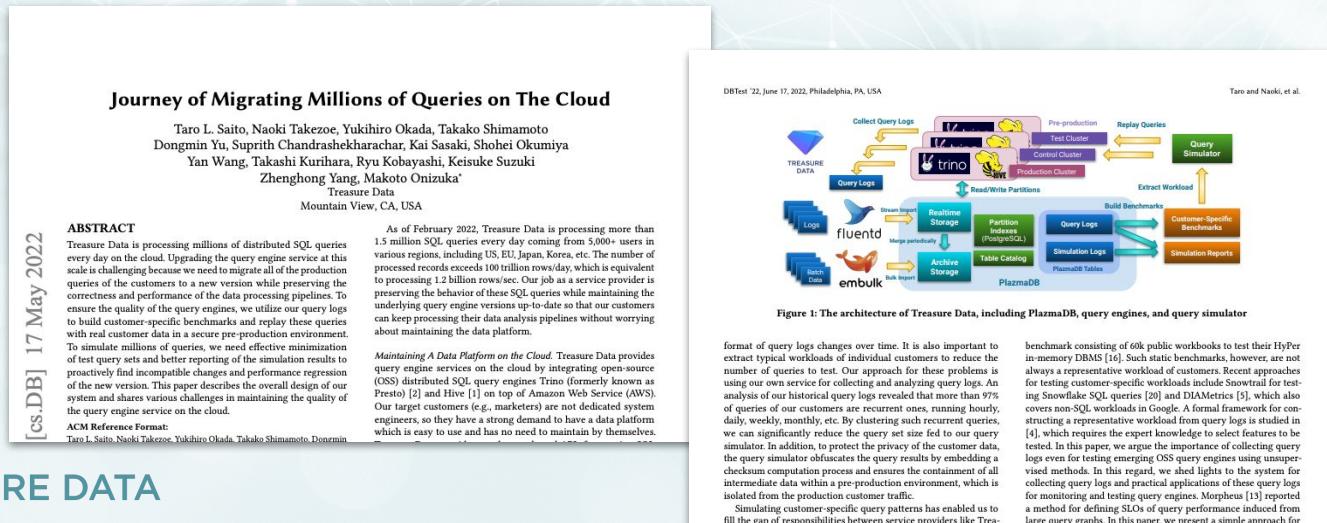
    > show leftjoin
    file1
    file2
    file3
    file4
    file5
    file6
    file7
    file8
    file9
    file10
    file11
    file12
    file13
    file14
    file15
    file16
    file17
    file18
    file19
    file20
    file21
    file22
    file23
    file24
    file25
    file26
    file27
    file28
    file29
    file30
    file31
    file32
    file33
    file34
    file35
    file36
    file37
    file38
    file39
    file40
    file41
    file42
    file43
    file44
    file45
    file46
    file47
    file48
    file49
    file50
    file51
    file52
    file53
    file54
    file55
    file56
    file57
    file58
    file59
    file60
    file61
    file62
    file63
    file64
    file65
    file66
    file67
    file68
    file69
    file70
    file71
    file72
    file73
    file74
    file75
    file76
    file77
    file78
    file79
    file80
    file81
    file82
    file83
    file84
    file85
    file86
    file87
    file88
    file89
    file90
    file91
    file92
    file93
    file94
    file95
    file96
    file97
    file98
    file99
    file100
    file101
    file102
    file103
    file104
    file105
    file106
    file107
    file108
    file109
    file110
    file111
    file112
    file113
    file114
    file115
    file116
    file117
    file118
    file119
    file120
    file121
    file122
    file123
    file124
    file125
    file126
    file127
    file128
    file129
    file130
    file131
    file132
    file133
    file134
    file135
    file136
    file137
    file138
    file139
    file140
    file141
    file142
    file143
    file144
    file145
    file146
    file147
    file148
    file149
    file150
    file151
    file152
    file153
    file154
    file155
    file156
    file157
    file158
    file159
    file160
    file161
    file162
    file163
    file164
    file165
    file166
    file167
    file168
    file169
    file170
    file171
    file172
    file173
    file174
    file175
    file176
    file177
    file178
    file179
    file180
    file181
    file182
    file183
    file184
    file185
    file186
    file187
    file188
    file189
    file190
    file191
    file192
    file193
    file194
    file195
    file196
    file197
    file198
    file199
    file200
    file201
    file202
    file203
    file204
    file205
    file206
    file207
    file208
    file209
    file210
    file211
    file212
    file213
    file214
    file215
    file216
    file217
    file218
    file219
    file220
    file221
    file222
    file223
    file224
    file225
    file226
    file227
    file228
    file229
    file230
    file231
    file232
    file233
    file234
    file235
    file236
    file237
    file238
    file239
    file240
    file241
    file242
    file243
    file244
    file245
    file246
    file247
    file248
    file249
    file250
    file251
    file252
    file253
    file254
    file255
    file256
    file257
    file258
    file259
    file260
    file261
    file262
    file263
    file264
    file265
    file266
    file267
    file268
    file269
    file270
    file271
    file272
    file273
    file274
    file275
    file276
    file277
    file278
    file279
    file280
    file281
    file282
    file283
    file284
    file285
    file286
    file287
    file288
    file289
    file290
    file291
    file292
    file293
    file294
    file295
    file296
    file297
    file298
    file299
    file300
    file301
    file302
    file303
    file304
    file305
    file306
    file307
    file308
    file309
    file310
    file311
    file312
    file313
    file314
    file315
    file316
    file317
    file318
    file319
    file320
    file321
    file322
    file323
    file324
    file325
    file326
    file327
    file328
    file329
    file330
    file331
    file332
    file333
    file334
    file335
    file336
    file337
    file338
    file339
    file340
    file341
    file342
    file343
    file344
    file345
    file346
    file347
    file348
    file349
    file350
    file351
    file352
    file353
    file354
    file355
    file356
    file357
    file358
    file359
    file360
    file361
    file362
    file363
    file364
    file365
    file366
    file367
    file368
    file369
    file370
    file371
    file372
    file373
    file374
    file375
    file376
    file377
    file378
    file379
    file380
    file381
    file382
    file383
    file384
    file385
    file386
    file387
    file388
    file389
    file390
    file391
    file392
    file393
    file394
    file395
    file396
    file397
    file398
    file399
    file400
    file401
    file402
    file403
    file404
    file405
    file406
    file407
    file408
    file409
    file410
    file411
    file412
    file413
    file414
    file415
    file416
    file417
    file418
    file419
    file420
    file421
    file422
    file423
    file424
    file425
    file426
    file427
    file428
    file429
    file430
    file431
    file432
    file433
    file434
    file435
    file436
    file437
    file438
    file439
    file440
    file441
    file442
    file443
    file444
    file445
    file446
    file447
    file448
    file449
    file450
    file451
    file452
    file453
    file454
    file455
    file456
    file457
    file458
    file459
    file460
    file461
    file462
    file463
    file464
    file465
    file466
    file467
    file468
    file469
    file470
    file471
    file472
    file473
    file474
    file475
    file476
    file477
    file478
    file479
    file480
    file481
    file482
    file483
    file484
    file485
    file486
    file487
    file488
    file489
    file490
    file491
    file492
    file493
    file494
    file495
    file496
    file497
    file498
    file499
    file500
    file501
    file502
    file503
    file504
    file505
    file506
    file507
    file508
    file509
    file510
    file511
    file512
    file513
    file514
    file515
    file516
    file517
    file518
    file519
    file520
    file521
    file522
    file523
    file524
    file525
    file526
    file527
    file528
    file529
    file530
    file531
    file532
    file533
    file534
    file535
    file536
    file537
    file538
    file539
    file540
    file541
    file542
    file543
    file544
    file545
    file546
    file547
    file548
    file549
    file550
    file551
    file552
    file553
    file554
    file555
    file556
    file557
    file558
    file559
    file560
    file561
    file562
    file563
    file564
    file565
    file566
    file567
    file568
    file569
    file570
    file571
    file572
    file573
    file574
    file575
    file576
    file577
    file578
    file579
    file580
    file581
    file582
    file583
    file584
    file585
    file586
    file587
    file588
    file589
    file590
    file591
    file592
    file593
    file594
    file595
    file596
    file597
    file598
    file599
    file600
    file601
    file602
    file603
    file604
    file605
    file606
    file607
    file608
    file609
    file610
    file611
    file612
    file613
    file614
    file615
    file616
    file617
    file618
    file619
    file620
    file621
    file622
    file623
    file624
    file625
    file626
    file627
    file628
    file629
    file630
    file631
    file632
    file633
    file634
    file635
    file636
    file637
    file638
    file639
    file640
    file641
    file642
    file643
    file644
    file645
    file646
    file647
    file648
    file649
    file650
    file651
    file652
    file653
    file654
    file655
    file656
    file657
    file658
    file659
    file660
    file661
    file662
    file663
    file664
    file665
    file666
    file667
    file668
    file669
    file670
    file671
    file672
    file673
    file674
    file675
    file676
    file677
    file678
    file679
    file680
    file681
    file682
    file683
    file684
    file685
    file686
    file687
    file688
    file689
    file690
    file691
    file692
    file693
    file694
    file695
    file696
    file697
    file698
    file699
    file700
    file701
    file702
    file703
    file704
    file705
    file706
    file707
    file708
    file709
    file710
    file711
    file712
    file713
    file714
    file715
    file716
    file717
    file718
    file719
    file720
    file721
    file722
    file723
    file724
    file725
    file726
    file727
    file728
    file729
    file730
    file731
    file732
    file733
    file734
    file735
    file736
    file737
    file738
    file739
    file740
    file741
    file742
    file743
    file744
    file745
    file746
    file747
    file748
    file749
    file750
    file751
    file752
    file753
    file754
    file755
    file756
    file757
    file758
    file759
    file760
    file761
    file762
    file763
    file764
    file765
    file766
    file767
    file768
    file769
    file770
    file771
    file772
    file773
    file774
    file775
    file776
    file777
    file778
    file779
    file780
    file781
    file782
    file783
    file784
    file785
    file786
    file787
    file788
    file789
    file790
    file791
    file792
    file793
    file794
    file795
    file796
    file797
    file798
    file799
    file800
    file801
    file802
    file803
    file804
    file805
    file806
    file807
    file808
    file809
    file8010
    file8011
    file8012
    file8013
    file8014
    file8015
    file8016
    file8017
    file8018
    file8019
    file8020
    file8021
    file8022
    file8023
    file8024
    file8025
    file8026
    file8027
    file8028
    file8029
    file8030
    file8031
    file8032
    file8033
    file8034
    file8035
    file8036
    file8037
    file8038
    file8039
    file8040
    file8041
    file8042
    file8043
    file8044
    file8045
    file8046
    file8047
    file8048
    file8049
    file8050
    file8051
    file8052
    file8053
    file8054
    file8055
    file8056
    file8057
    file8058
    file8059
    file8060
    file8061
    file8062
    file8063
    file8064
    file8065
    file8066
    file8067
    file8068
    file8069
    file8070
    file8071
    file8072
    file8073
    file8074
    file8075
    file8076
    file8077
    file8078
    file8079
    file8080
    file8081
    file8082
    file8083
    file8084
    file8085
    file8086
    file8087
    file8088
    file8089
    file8090
    file8091
    file8092
    file8093
    file8094
    file8095
    file8096
    file8097
    file8098
    file8099
    file80100
    file80101
    file80102
    file80103
    file80104
    file80105
    file80106
    file80107
    file80108
    file80109
    file80110
    file80111
    file80112
    file80113
    file80114
    file80115
    file80116
    file80117
    file80118
    file80119
    file80120
    file80121
    file80122
    file80123
    file80124
    file80125
    file80126
    file80127
    file80128
    file80129
    file80130
    file80131
    file80132
    file80133
    file80134
    file80135
    file80136
    file80137
    file80138
    file80139
    file80140
    file80141
    file80142
    file80143
    file80144
    file80145
    file80146
    file80147
    file80148
    file80149
    file80150
    file80151
    file80152
    file80153
    file80154
    file80155
    file80156
    file80157
    file80158
    file80159
    file80160
    file80161
    file80162
    file80163
    file80164
    file80165
    file80166
    file80167
    file80168
    file80169
    file80170
    file80171
    file80172
    file80173
    file80174
    file80175
    file80176
    file80177
    file80178
    file80179
    file80180
    file80181
    file80182
    file80183
    file80184
    file80185
    file80186
    file80187
    file80188
    file80189
    file80190
    file80191
    file80192
    file80193
    file80194
    file80195
    file80196
    file80197
    file80198
    file80199
    file80200
    file80201
    file80202
    file80203
    file80204
    file80205
    file80206
    file80207
    file80208
    file80209
    file80210
    file80211
    file80212
    file80213
    file80214
    file80215
    file80216
    file80217
    file80218
    file80219
    file80220
    file80221
    file80222
    file80223
    file80224
    file80225
    file80226
    file80227
    file80228
    file80229
    file80230
    file80231
    file80232
    file80233
    file80234
    file80235
    file80236
    file80237
    file80238
    file80239
    file80240
    file80241
    file80242
    file80243
    file80244
    file80245
    file80246
    file80247
    file80248
    file80249
    file80250
    file80251
    file80252
    file80253
    file80254
    file80255
    file80256
    file80257
    file80258
    file80259
    file80260
    file80261
    file80262
    file80263
    file80264
    file80265
    file80266
    file80267
    file80268
    file80269
    file80270
    file80271
    file80272
    file80273
    file80274
    file80275
    file80276
    file80277
    file80278
    file80279
    file80280
    file80281
    file80282
    file80283
    file80284
    file80285
    file80286
    file80287
    file80288
    file80289
    file80290
    file80291
    file80292
    file80293
    file80294
    file80295
    file80296
    file80297
    file80298
    file80299
    file80300
    file80301
    file80302
    file80303
    file80304
    file80305
    file80306
    file80307
    file80308
    file80309
    file80310
    file80311
    file80312
    file80313
    file80314
    file80315
    file80316
    file80317
    file80318
    file80319
    file80320
    file80321
    file80322
    file80323
    file80324
    file80325
    file80326
    file80327
    file80328
    file80329
    file80330
    file80331
    file80332
    file80333
    file80334
    file80335
    file80336
    file80337
    file80338
    file80339
    file80340
    file80341
    file80342
    file80343
    file80344
    file80345
    file80346
    file80347
    file80348
    file80349
    file80350
    file80351
    file80352
    file80353
    file80354
    file80355
    file80356
    file80357
    file80358
    file80359
    file80360
    file80361
    file80362
    file80363
    file80364
    file80365
    file80366
    file80367
    file80368
    file80369
    file80370
    file80371
    file80372
    file80373
    file80374
    file80375
    file80376
    file80377
    file80378
    file80379
    file80380
    file80381
    file80382
    file80383
    file80384
    file80385
    file80386
    file80387
    file80388
    file80389
    file80390
    file80391
    file80392
    file80393
    file80394
    file80395
    file80396
    file80397
    file80398
    file80399
    file80400
    file80401
    file80402
    file80403
    file80404
    file80405
    file80406
    file80407
    file80408
    file80409
    file80410
    file80411
    file80412
    file80413
    file80414
    file80415
    file80416
    file80417
    file80418
    file80419
    file80420
    file80421
    file80422
    file80423
    file80424
    file80425
    file80426
    file80427
    file80428
    file80429
    file80430
    file80431
    file80432
    file80433
    file80434
    file80435
    file80436
    file80437
    file80438
    file80439
    file80440
    file80441
    file80442
    file80443
    file80444
    file80445
    file80446
    file80447
    file80448
    file80449
    file80450
    file80451
    file80452
    file80453
    file80454
    file80455
    file80456
    file80457
    file80458
    file80459
    file80460
    file80461
    file80462
    file80463
    file80464
    file80465
    file80466
    file80467
    file80468
    file80469
    file80470
    file80471
    file80472
    file80473
    file80474
    file80475
    file80476
    file80477
    file80478
    file80479
    file80480
    file80481
    file80482
    file80483
    file80484
    file80485
    file80486
    file80487
    file80488
    file80489
    file80490
    file80491
    file80492
    file80493
    file80494
    file80495
    file80496
    file80497
    file80498
    file80499
    file80500
    file80501
    file80502
    file80503
    file80504
    file80505
    file80506
    file80507
    file80508
    file80509
    file80510
    file80511
    file80512
    file80513
    file80514
    file80515
    file80516
    file80517
    file80518
    file80519
    file80520
    file80521
    file80522
    file80523
    file80524
    file80525
    file80526
    file80527
    file80528
    file80529
    file80530
    file80531
    file80532
    file80533
    file80534
    file80535
    file80536
    file80537
    file80538
    file80539
    file80540
    file80541
    file80542
    file80543
    file80544
    file80545
    file80546
    file80547
    file80548
    file80549
    file80550
    file80551
    file80552
    file80553
    file80554
    file80555
    file80556
    file80557
    file80558
    file80559
    file80560
    file80561
    file80562
    file80563
    file80564
    file80565
    file80566
    file80567
    file80568
    file80569
    file80570
    file80571
    file80572
    file80573
    file80574
    file80575
    file80576
    file80577
    file80578
    file80579
    file80580
    file80581
    file80582
    file80583
    file80584
    file80585
    file80586
    file80587
    file80588
    file80589
    file80590
    file80591
    file80592
    file80593
    file80594
    file80595
    file80596
    file80597
    file80598
    file80599
    file80600
    file80601
    file80602
    file80603
    file80604
    file80605
    file80606
    file80607
    file80608
    file80609
    file80610
    file80611
    file80612
    file80613
    file80614
    file80615
    file80616
    file80617
    file80618
    file80619
    file80620
    file80621
    file80622
    file80623
    file80624
    file80625
    file80626
    file80627
    file80628
    file80629
    file80630
    file80631
    file80632
    file80633
    file80634
    file80635
    file80636
    file80637
    file80638
    file80639
    file80640
    file80641
    file80642
    file80643
    file80644
    file80645
    file80646
    file80647
    file80648
    file80649
    file80650
    file80651
    file80652
    file80653
    file80654
    file80655
    file80656
    file80657
    file80658
    file80659
    file80660
    file80661
    file80662
    file80663
    file80664
    file80665
    file80666
    file80667
    file80668
    file80669
    file80670
    file80671
    file80672
    file80673
    file80674
    file80675
    file80676
    file80677
    file80678
    file80679
    file80680
    file80681
    file80682
    file80683
    file80684
    file80685
    file80686
    file80687
    file80688
    file80689
    file80690
    file80691
    file80692
    file80693
    file80694
    file80695
    file80696
    file80697
    file80698
    file80699
    file80700
    file80701
    file80702
    file80703
    file80704
    file80705
    file80706
    file80707
    file80708
    file80709
    file80710
    file80711
    file80712
    file80713
    file80714
    file80715
    file80716
    file80717
    file80718
    file80719
    file80720
    file80721
    file80722
    file80723
    file80724
    file80725
    file80726
    file80727
    file80728
    file80729
    file80730
    file80731
    file80732
    file80733
    file80734
    file80735
    file80736
    file80737
    file80738
    file80739
    file80740
    file80741
    file80742
    file80743
    file80744
    file80745
    file80746
    file80747
    file80748
    file80749
    file80750
    file80751
    file80752
    file80753
    file80754
    file80755
    file80756
    file80757
    file80758
    file80759
    file80760
    file80761
    file80762
    file80763
    file80764
    file80765
    file80766
    file80767
    file80768
    file80769
    file80770
    file80771
    file80772
    file80773
    file80774
    file80775
    file80776
    file80777
    file80778
    file80779
    file80780
    file80781
    file80782
    file80783
    file80784
    file80785
    file80786
    file80787
    file80788
    file80789
    file80790
    file80791
    file80792
    file80793
    file80794
    file80795
    file80796
    file80797
    file80798
    file80799
    file80800
    file80801
    file80802
    file80803
    file80804
    file80805
    file80806
    file80807
    file80808
    file80809
    file808010
    file808011
    file808012
    file808013
    file808014
    file808015
    file808016
    file808017
    file808018
    file808019
    file808020
    file808021
    file808022
    file808023
    file808024
    file808025
    file808026
    file808027
    file808028
    file808029
    file808030
    file808031
    file808032
    file808033
    file808034
    file808035
    file808036
    file808037
    file808038
    file808039
    file808040
    file808041
    file808042
    file808043
    file808044
```

Trino at Treasure Data (2024)

- Treasure Data has been operating Trino (formerly Presto) as a service since 2014 (10th anniversary!)
 - 3+ million Trino SQL queries processed / day
 - 400+ trillion rows processed / day
 - 3+ billion S3 GET requests / day
 - Reduced from 10 billion requests / day (2023) by partition optimization
- In 2024, we completed the customer traffic migration from Presto (350) to Trino
 - See our migration/test methods in DBTest 2022 paper
- A lot of challenges in managing SQL and helping users (or LLM) write efficient SQL queries



TREASURE
DATA



Journey of Migrating Millions of Queries on The Cloud

Taro L. Saito, Naoki Takezoe, Yukihiro Okada, Takako Shimamoto
Dongmin Yu, Supriith Chandrashekharachar, Kai Sasaki, Shohei Okumura
Yan Wang, Takashi Kurihara, Ryu Kobayashi, Keisuke Suzuki
Zhenghong Yang, Makoto Onizuka^{*}
Treasure Data
Mountain View, CA, USA

ABSTRACT

Treasure Data is processing millions of distributed SQL queries every day. Updating the query engine service and scale it is challenging because we need to migrate all of the production queries of the customers to a new version while preserving the correctness and performance of the data processing pipelines. To ensure the quality of the query engines, we utilize our query logs to build customer-specific benchmarks and replay these queries with real customer data in a secure pre-production environment. To simulate millions of queries, we need effective minimization of test query sets and better reporting of the simulation results to proactively find incompatible changes and performance regression of the new version. This paper describes the overall design of our system and shares various challenges in maintaining the quality of the query engine service on the cloud.

ACM Reference Format:

Taro L. Saito, Naoki Takezoe, Yukihiro Okada, Takako Shimamoto, Dongmin

[cs DB] 17 May 2022

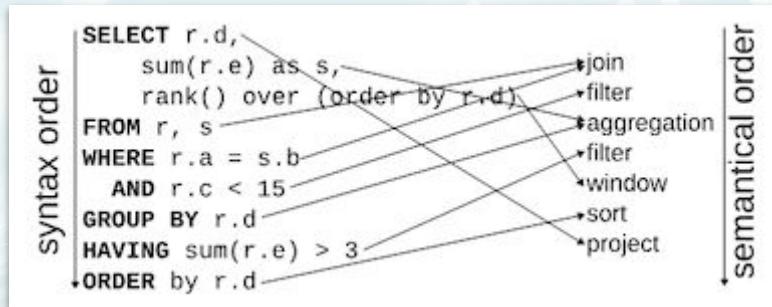
format of query logs changes over time. It is also important to extract typical workloads of individual customers to reduce the number of queries to test. Our approach for these problems is to use a query simulator that replay the historical query logs. An analysis of our historical query logs revealed that more than 70% of queries of our customers are recurrent ones, running hourly, daily, weekly, monthly, etc. By clustering such recurrent queries, we can significantly reduce the query set size fed to our query simulator. In addition, to protect the privacy of the customer data, the query simulator obfuscates the query results by embedding a checksum computation process and ensures the containment of all intermediate data within a pre-production environment, which is isolated from the production customer traffic.

Simulating customer-specific query patterns has enabled us to fill the gap of responsibilities between service providers like Treas-

ure Data and customers. We have conducted a series of experiments to validate the effectiveness of our approach. The results show that our approach can significantly reduce the number of queries to test while maintaining the quality of the query engines. In this paper, we argue the importance of collecting query logs even for testing emerging OSS query engines using unsupervised methods. In this regard, we shed lights to the system for collecting query logs and practical applications of these query logs for monitoring and testing query engines. Morpheus [13] reported a method for defining SLOs of query performance induced from large query graphs. In this paper, we present a simple approach for

What's Wrong with SQL?

- The syntactic order of SQL doesn't match the actual data flow
 - [A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language \(CIDR 24\)](#)
 - Even SQL experts find it challenging to debug nested queries



- Lacks essential software engineering features for managing many queries
 - No built-in support for reusing queries
 - No entry points for multi-query optimization
 - e.g., incremental processing and pipeline execution like dbt
 - No built-in debugging or testing capabilities

Wvlet: Modernizing 50-Year-Old SQL

- SQL has proven to be effective and useful
 - Its natural-language based syntax has gained widespread adoption
- Redesigned SQL to be more intuitive and functional
- Wvlet: a new flow-style query language for weaving data
 - Pronounced as weave-let
 - Queries start with “from” for intuitive data flow
- Functional
 - Wvlet queries are reusable and composable like functions
- Cross SQL engine support
 - Generates SQL for Trino, Hive, DuckDB, etc.
- Try Wvlet on your web browser <https://wvlet.org/>
 - No installation is required

```
wv> -- Scan a data file
      from 'lineitem.parquet'
      -- Apply a filtering condition
      where l_quantity > 10.0
      -- Aggregate by columns
      group by l_returnflag, l_linenumber
      -- Compute aggregated expressions
      agg
          l_quantity.sum as qty,
          l_extendedprice.sum as price,
          l_discount.avg.round(4) as discount_avg,
      -- Sort results
      order by qty desc
      -- Remove an unnecessary column
      exclude discount_avg
      -- Save query with an alias
      select as stats
;
```

l_returnflag string	l_linenumber string	qty decimal(38,2)	price decimal(38,2)
N	0	732001.00	1026373433.61
R	F	365309.00	511907681.35
A	F	364109.00	509533926.17
N	F	8642.00	11948259.73

```
wv> [
```

Command-line editor of Wvlet (wv)

Wvlet Playground wvlet.org

The screenshot shows the Wvlet Playground interface with the following components:

- Editor:** Displays the Wvlet query code for "00_sample.wv".
- Compiled SQL:** Shows the generated SQL code.
- Preview:** A table showing the results of the query.
- Query Result:** A summary of the query results.

Wvlet Query:

```
1 -- Scan a table
2 from lineitem
3 -- Add filtering condition
4 where l_quantity > 0.0
5 -- Grouping by keys
6 group by l_returnflag, l_linenstatus
7 -- Add aggregation expressions
8 agg
9   l_quantity.sum as sum_qty,
10  l_extendedprice.sum as sum_ext_price
11 -- Remove unnecessary column
12 exclude l_returnflag
13 -- Sort
14 order by sum_qty desc
```

Compiled SQL:

```
1 -- wvlet version=2024.9.13
+33-7bdc7959, src=01JEP56BY3DWGB0N50ZBFAX934.wv:14
2 select * from
3   (select l_linenstatus, sum_qty, sum_ext_price from
4     (select l_returnflag as l_returnflag, l_linenstatus as
5       l_linenstatus, sum(l_quantity) as sum_qty, sum
6       (l_extendedprice) as sum_ext_price
7     from
8       (select * from lineitem
9        where l_quantity > 0.0)
10      group by l_returnflag, l_linenstatus))
11 order by sum_qty desc
```

Preview:

l_linenstatus	sum_qty	sum_ext_price
0	76525100	107286230210
F	38144900	53459444535
F	38045600	53234821165
F	897100	1238480137

Query Result:

4 rows



Wvlet: Architecture

- From query (.wv) files, Wvlet compiler produces logical plans, execution plans, and SQL statements
- Logical Plans
 - Tree-representations of relational operators (e.g., scan, filter, projection)
 - Sort(Join(Filter(TableScan(...))), TableScan(...)))
- Execution Plans
 - A sequence of steps to execute SQL and other programs
 - ExecutePlan(ExecuteSQL(query), ExecuteTest(expr), ExecuteCommand(expr), ...)



Wvlet: Analyze As You Write

- Flow-style queries
 - Each line is a single operation
- Peek the data and schema at any point
 - A subquery becomes a range of lines in the query text
- Easy to extend
 - Queries can be reused for further analysis



Related Work: GoogleSQL Pipe Syntax (2024)

- SQL Has Problems. We Can Fix Them: Pipe Syntax
In SQL (VLDB 2024)
- Extended SQL syntax with pipe operator (|>)
 - Available in ZetaSQL, an open-source SQL parser, used in Google products, e.g., BigQuery, F1, etc.
- Other flow-style languages:
 - PRQL, Microsoft Kusto, DryadLINQ, etc.

SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL

Jeff Shute, Shannon Bales, Matthew Brown, Jean-Daniel Browne, Brandon Dolphin
Google, Inc. Google, Inc. Google, Inc. Google, Inc. Google, Inc.

Romit Kudtarkar, Andrey Litvinov, Jingchi Ma, John Morcos, Michael Shen
Google, Inc. Google, Inc. Google, Inc. Google, Inc. Google, Inc.

David Wilhite, Xi Wu, Lulan Yu
Google, Inc. Google, Inc. Google, Inc.

sql-pipes-paper@google.com

ABSTRACT
SQL has been extremely successful as the de facto standard language for working with data. Virtually all mainstream database-like systems use SQL as their primary query language. But SQL is an old language with significant design problems, making it difficult to learn, difficult to use, and difficult to extend. Many have observed these challenges with SQL, and proposed solutions involving new languages. New language adoption is a significant obstacle for users, and most of the potential replacements have been successful enough to drop SQL.

In GoogleSQL, we've taken a different approach - solving SQL's problems by extending SQL. Inspired by a pattern that works well in other modern data languages, we added piped data flow syntax

SQL. Migrating away from existing SQL ecosystems is expensive and generally unpalatable for users.

This paper presents a different approach. After describing the most critical problems with the SQL language, we present a solution - adding pipe-structured data flow syntax to SQL. This makes SQL more flexible, extensible and easy to use. This paradigm works well in other languages like Kusto's KQL[5] and in APIs like Apache Beam[1]. We show how syntax can be added to SQL too, without removing anything, and while maintaining full backwards compatibility and interoperability.

In SQL, the standard clauses occur in one rigidly defined order. Expressing anything else requires subqueries or other workarounds. With pipe syntax, operations can be composed arbitrarily, in any

```
SELECT c_count, COUNT(*) AS custdist
FROM
  ( SELECT c_custkey, COUNT(o_orderkey) c_count
    FROM customer
    LEFT OUTER JOIN orders ON c_custkey = o_custkey
      AND o_comment NOT LIKE '%unusual%packages%'
    GROUP BY c_custkey
  ) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```

SQL



```
FROM customer
|> LEFT OUTER JOIN orders ON c_custkey = o_custkey
  AND o_comment NOT LIKE '%unusual%packages%'
|> AGGREGATE COUNT(o_orderkey) c_count
  GROUP BY c_custkey
|> AGGREGATE COUNT(*) AS custdist
  GROUP BY c_count
|> ORDER BY custdist DESC, c_count DESC;
```

SQL + Pipe Syntax

Wvlet: Relational Operators ([online reference](#))

- In Wvlet, all keywords must be lower-case letters
 - For consistency and reducing implementation efforts
- Same operators with SQL:
 - **from**, **select**, **join**, **where**, **order by**, **limit**, etc.
- Aggregation in Wvlet
 - **group by** k1, k2, ... **agg** ...
 - **agg** (aggregation expr), ..
 - Report group-by keys and aggregation expressions: k1, k2, ..., expr1, expr2, ...
 - **group by** k1, k2, ... **where** ...
 - equivalent to group by ... having ...

```
wv> from lineitem  
      group by l_shipmode  
      agg _.count, l_quantity.sum;
```

l_shipmode	count_star()	sum(l_quantity)
string	long	decimal(38,2)
FOB	8641	219565.00
SHIP	8482	217969.00
REG AIR	8616	219015.00
TRUCK	8710	223909.00
AIR	8491	216331.00
MAIL	8669	221528.00
RAIL	8566	217810.00

7 rows

Aggregation Query

```
wv> from ...          -- Scan the input data  
      where ...        -- Apply filtering conditions  
      where ...        -- [optional] Apply more filtering conditions  
      add ... as ...    -- Add new columns  
      transform ...     -- Transform a subset of columns  
      group by ...      -- Grouping rows by the given columns  
      agg ...           -- Add group aggregation expressions,  
                        -- (e.g., _.count, _.sum)  
      where ...         -- Apply filtering conditions for groups  
                        -- (e.g., HAVING clause in SQL)  
      exclude ...       -- Remove columns from the output  
      shift ...          -- Shift the column position to the left  
      select ...         -- Select columns to output  
      order by ...       -- Sort the rows by the given columns  
      limit ...          -- Limit the number of rows to output
```

Basic Flow of Wvlet Queries

Wvlet: Column At A Time Manipulation

- For minimizing your typing effort
- **add (expr) as ...**
 - Add a new column
- **rename (column) as (new name)**
- **exclude (column), ...**
- **shift (to left/right)? column,**
 - Reorder columns for readability
- Changing a single column or its order in SQL is not easy as we need to enumerate all columns

```
wv> from lineitem  
  where l_returnflag = 'N'  
  add s"${l_shipmode}:${l_shipinstruct}" as label  
  shift label, l_returnflag  
  limit 5;
```

label string	l_returnflag string	l_shipmode string	l_shipinstruct string	l_qu decim
TRUCK:DELIVER IN PERSON	N	TRUCK	DELIVER IN PERSON	
MAIL:TAKE BACK RETURN	N	MAIL	TAKE BACK RETURN	
REG AIR:TAKE BACK RETURN	N	REG AIR	TAKE BACK RETURN	
AIR:NONE	N	AIR	NONE	
FOB:NONE	N	FOB	NONE	

5 rows

Wvlet

```
-- wvlet version:2024.9.9, src:01JBYY824NEXQ9BTC9RKYERFG.wv  
select * from  
  (select label, l_returnflag, l_shipmode, l_shipinstruct, l_quantity, l_tax, l_shipdate, l_suppkey, l_orderkey, l_comment, l_linenumber, l_receiptdate, l_linestatus, l_extendedprice, l_discount, l_commitdate from  
    (select *, concat(concat(concat(concat('' , l_shipmode) , ':') , l_shipinstruct) , '') as label from  
      (select * from lineitem  
       where l_returnflag = 'N')))  
limit 5 - (QueryExecutor.scala:319)
```

SQL



Wvlet: Additional Relational Operators

- Refined from SQL
 - **concat** = UNION ALL in SQL
 - Most of SQL users wrongly use UNION (duplicate elimination) where UNION ALL (concatenate two relations) is appropriate.
 - **dedup** = select distinct * (= duplicate elimination)
- Utility operators
 - **transform**: Update only subset of columns
 - **sample n**
 - **pivot**
 - Transform column values into individual columns
 - [trino#1206](#) (Remains open since 2019)
 - Wvlet can run multiple SQL queries, so it's relatively easy to implement static/dynamic pivot

```
wv> from lineitem  
      where l_returnflag = 'N'  
      limit 3  
  concat {  
    from lineitem  
    where l_returnflag = 'A'  
    limit 3  
  }  
  shift l_returnflag
```

Wvlet

```
-- wvlet version:2024.9.9, src:01JBYX6C240NPZZ9ZR3Y0T9YA9.wv  
select l_returnflag, l_shipmode, l_shipinstruct, l_quantity, l_tax, l_  
shipdate, l_suppkey, l_orderkey, l_comment, l_linenumber, l_extendedprice, l_partkey, l_discount, l_commitdate  
from  
(  
  select * from  
  (select * from lineitem  
  where l_returnflag = 'N')  
  limit 3)  
union all  
  
(  
  select * from  
  (select * from lineitem  
  where l_returnflag = 'A')  
  limit 3)) - (QueryExecutor.scala:319)
```

SQL

Wvlet: Update Statements

- **save as**
 - A shorthand notation for:
 - DROP TABLE IF EXISTS tbl; CREATE TABLE tbl AS SELECT ...
 - Most frequently used pattern in Treasure Data
 - CREATE OR REPLACE TABLE AS SELECT
 - Available in DuckDB, [Trino 431](#) (Since Oct 2023)
 - For DuckDB backend, you can save query results as Parquet/JSON files
- **append to**
 - Almost same with INSERT INTO, but it clarifies append-only semantics (no overwrites)
 - Useful when using versioned tables like Iceberg/Delta Lake
- **delete**
 - Delete selected rows from the table

```
wv> from lineitem  
| where l_returnflag = 'N'  
| limit 10  
| save as tmp_table;
```

Wvlet

```
Executing SQL:  
create or replace table tmp_table as  
-- wvlet version:2024.9.9, src:01JBYYKEV8CK8HTZEBZGAMYVZB.wv  
select * from  
(select * from lineitem  
where l_returnflag = 'N')  
limit 10 - (QueryExecutor.scala:173)
```

SQL



Function Chaining via Dot Operator

- In SQL, every function is global
 - SUM(x), AVG(x), COUNT(*), ROUND(x, 1), ...
 - A lot of cursor movements are necessary to apply functions in SQL
- Wvlet supports chaining functions with dot operator as in modern programming languages:
 - e.g., x.sum, x.avg.round(2), _.count
 - _ (underscore) refers to the output from the previous operator

```
select
    l_shipmode,
    round(avg(l_quantity), 1),
    sum(l_quantity),
    count(*)
from lineitem
group by l_shipmode
;■
```

SQL

```
wv> from lineitem
      group by l_shipmode
      agg
          l_quantity.avg.round(1),
          l_quantity.sum,
          _.count,
      ;
```

l_shipmode string	round(avg(l_quantity), 1) double	sum(l_quantity) decimal(38,2)	count_star() long
REG AIR	25.4	219015.00	8616
FOB	25.4	219565.00	8641
SHIP	25.7	217969.00	8482
MAIL	25.6	221528.00	8669
RAIL	25.4	217810.00	8566
AIR	25.5	216331.00	8491
TRUCK	25.7	223909.00	8710

7 rows

Wvlet

Wvlet Query = A Function for Building A Data Model

- In Wvlet, a query defined as a data model that can be reused at ease (Query Templates)
- Composable
 - You can add more relational operators (e.g., where, join) to data models
- Reusable
 - Models accept user parameters (= function arguments)

```
wv> -- define a reusable query for analyzing cost
| model service_cost =
|   from cost_components
|   where time.within('-3M/now')
|   where aws_account_owner != 'security'
|   add service_name = coalesce(user_service, cost_category)
|   add td_time_string(time, 'M!') as month
|   pivot on month in ('2024-06', '2024-07', '2024-08')
|   group by service_name
|   agg cost.sum.round(1)
| end;
```

Reuse Queries



wv> from service_cost where service_name like 'trino%';	<table border="1"><thead><tr><th>service_name</th><th>2024-06</th><th>2024-07</th><th>2024-08</th></tr><tr><th>string</th><th>double</th><th>double</th><th>double</th></tr></thead><tbody><tr><td>trino-coordinator</td><td>3415.3</td><td>3983.4</td><td>4402.7</td></tr><tr><td>trino-worker</td><td>33776.7</td><td>58363.2</td><td>78516.8</td></tr></tbody></table> <p>2 rows</p>	service_name	2024-06	2024-07	2024-08	string	double	double	double	trino-coordinator	3415.3	3983.4	4402.7	trino-worker	33776.7	58363.2	78516.8
service_name	2024-06	2024-07	2024-08														
string	double	double	double														
trino-coordinator	3415.3	3983.4	4402.7														
trino-worker	33776.7	58363.2	78516.8														
wv> from service_cost where service_name like 'hive%';	<table border="1"><thead><tr><th>service_name</th><th>2024-06</th><th>2024-07</th><th>2024-08</th></tr><tr><th>string</th><th>double</th><th>double</th><th>double</th></tr></thead><tbody><tr><td>hivep</td><td>174.9</td><td>164.4</td><td>188.4</td></tr><tr><td>hive-query-simulator</td><td>39.6</td><td>26.9</td><td>28.6</td></tr></tbody></table> <p>2 rows</p>	service_name	2024-06	2024-07	2024-08	string	double	double	double	hivep	174.9	164.4	188.4	hive-query-simulator	39.6	26.9	28.6
service_name	2024-06	2024-07	2024-08														
string	double	double	double														
hivep	174.9	164.4	188.4														
hive-query-simulator	39.6	26.9	28.6														

Testing Queries

- Wvlet Test Syntax
 - Useful for verifying resulting schema and results
- Wvlet is tested with Wvlet
 - Wvlet spec queries
 - Covers all 22 TPC-H queries

wvlet / spec / basic / ▾

⌚ xerial editor (feature #265): show query command (#334) ✓

Name

- 📁 ..
- 📁 catalog
- 📁 duckdb
- 📁 update
- 📄 agg_rel.wv
- 📄 agg_tpe_resolve.wv
- 📄 array_value.wv
- 📄 backquote-interpolation.wv
- 📄 backquote.wv
- 📄 books.json
- 📄 concat.wv
- 📄 debug-save.wv
- 📄 debug.wv

wvlet / spec / tpch / ▾

⌚ xerial lang (feature): test syntax as a simple language

Name

- 📁 ..
- 📁 data
- 📁 test
- 📄 q1.wv
- 📄 q10.wv
- 📄 q11.wv
- 📄 q12.wv
- 📄 q13.wv
- 📄 q14.wv
- 📄 q15.wv

person.json

```
[{"id":1, "name": "alice", "age": 10}, {"id":2, "name": "bob", "age": 24}, {"id":3, "name": "clark", "age": 40}]
```

from 'person.json'
-- Test the query size and schema
test _.size should be 3
test _.columns should be ['id', 'name', 'age']
test _.columns should contain 'name'
test _.columns should not contain 'address'

-- Test the query output. Surrounding white spaces will be trimmed:
test _.output should be """

id	name	age
long	string	long
1	alice	10
2	bob	24
3	clark	40

3 rows

....

Debugging Queries

- Debug operator can be used for checking intermediate query results
- ExecutionPlanner generates multiple execution paths for debug and regular query evaluation
 - Debug path: Query before debug statement -> debug query
 - Regular path: Query without debug statement

```
[{"id":1, "name": "alice", "age": 10},  
 {"id":2, "name": "bob", "age": 24},  
 {"id":3, "name": "clark", "age": 40}]
```

```
wv> from 'person.json'  
debug {  
    where id = 1  
    save as debug_result_0123  
}  
where id = 2  
  
from debug_result_0123  
test _.size should be 1  
test _.rows should be [[1, "alice", 10]]
```

debug (line:9): test _.rows should be [[1, "alice", 10]]

id long	name string	age long
2	bob	24
1 rows		

Regular Query Result

id long	name string	age long
1	alice	10
1 rows		

Debug Query Result

[ok]: 1 was equal to 1 (01JAXGC09FFAZ0ZNHAWWZSBJ4.wv:8:13)

[ok]: [[1, alice, 10]] was equal to [[1, alice, 10]] (01JAXGC09FFAZ0ZNHAWWZSBJ4.wv:9:13)

Test Result



Wvlet Internals: Compiler

- Compiler phases: Parser -> SymbolLabeler -> TypeResolver -> Rewriter -> ExecutionPlanner -> GenSQL
 - CompilationUnit holds the source text, untyped logical plan, typed logical plan, execution plan, etc.
- With [AirSpec](#) testing library, you can adjust the log levels of individual components
 - L (class name pattern)=(log level) option

```
wv> from 'books.json'  
| group by author  
| agg _.count;
```

Query (.wv)



```
sbt:wvlet> ~runner/testOnly *BasicSpec -- -l debug -L *Scanner=debug -L *GenSQL=trace agg_rel  
[info] compiling 1 Scala source to /Users/leo/work/git/wvlet/wvlet-api/.jvm/target/scala-3.3.4/classes ...  
BasicSpec:  
2024-10-23 13:29:36.786-0700 debug [WvletScanner] Indented(0,null) [ 0: 4]      FROM: from - (WvletScanner.scala:221)  
2024-10-23 13:29:36.803-0700 debug [WvletScanner] Indented(0,null) [ 5:12] STRING_LITERAL: books.json - (WvletScanner.scala:221)  
2024-10-23 13:29:36.804-0700 debug [WvletScanner] Indented(0,null) [ 18: 5] GROUP: group - (WvletScanner.scala:221)  
2024-10-23 13:29:36.804-0700 debug [WvletScanner] Indented(0,null) [ 24: 2] BY: by - (WvletScanner.scala:221)  
2024-10-23 13:29:36.804-0700 debug [WvletScanner] Indented(0,null) [ 27: 6] IDENTIFIER: author - (WvletScanner.scala:221)  
2024-10-23 13:29:36.805-0700 debug [WvletScanner] Indented(0,null) [ 34: 3] AGG: agg - (WvletScanner.scala:221)  
2024-10-23 13:29:36.805-0700 debug [WvletScanner] Indented(0,null) [ 38: 1] UNDERSCORE: _ - (WvletScanner.scala:221)  
2024-10-23 13:29:36.805-0700 debug [WvletScanner] Indented(0,null) [ 39: 1] DOT: . - (WvletScanner.scala:221)  
2024-10-23 13:29:38.367-0700 trace [GenSQL]  
[plan] runner / Test / testOnly 1s  
[Agg [34..45]] <:t167(author:string, _:array_t123(id:string, title:string, author:string, year:long, genre:string))> => <:t168(author:  
string, <empty>:???)>  
  - author:string := author  
  - <empty>:?? := sql"count(*)"  
  [GroupBy [18..33]] <:t123(id:string, title:string, author:string, year:long, genre:string)> => <:t167(author:string, _:array(_t123(id:  
string, title:string, author:string, year:long, genre:string)))>  
    - author  
    [JSONFileScan [5..17]] => <:t123(id:string, title:string, author:string, year:long, genre:string)>  
[SQL]  
-- wvlet version:2024.9.8+1-c700bcfa, src:agg_rel.wv  
select author as author, count(*)  
from 'spec/basic/books.json'  
group by author - (GenSQL.scala:87)  
2024-10-23 13:29:38.422-0700 debug [BasicSpec]
```

Generated SQL

LogicalPlan
(After typing)

Query Result

author string	count_star0 long
J.D. Salinger	2
F. Scott Fitzgerald	2
Harper Lee	2
John Steinbeck	1
4 rows	

- (SpecRunner.scala:52)



TREASURE DATA

Wvlet Development Roadmap

- Roadmap is maintained at [GitHub Project](#) page.
 - Planning milestones for about every 3 months
 - Versions will be YYYY.(milestone month).(patch) (See [#170](#) for the versioning scheme)
- Release 2024.9
 - Flow-style query language design and compiler
 - Including scanner, parser, typer, tree rewrite framework, execution planner, SQL generator, etc.
 - DBMS Connector (DuckDB, Trino)
 - wv: Interactive command-line editor (REPL)
 - Installer (Homebrew)
 - Web UI, Playground (Monaco Editor with DuckDB-Wasm)
- Release 2025.1
 - Model management
 - Generate dependent model materialization plan, like dbt
 - GitHub integration
 - Compiler plugins for advance optimization
 - Incremental processing, query fusion, etc.
 - Language SDKs
 - Python, Rust, C/C++, Java, etc.
 - SQL to Wvlet converter



Summary

- Designed Wvlet, a new open-source flow-style query language
- Addresses challenges in 50-year-old design of SQL
 - Leverage good sides of SQL
 - natural-language like syntax
 - Intuitive syntax for data flow
 - Reusability
 - Extensibility
- Website: <https://wvlet.org/>
- GitHub: <https://github.com/wvlet/wvlet>

The screenshot shows the Wvlet website interface. At the top, there's a navigation bar with links to Documentation, Query Syntax, Playground, Blog, and Release Notes. On the right side of the header are links to Roadmap, GitHub, and a search bar.

The main content area features a code editor with a query:Wvlet is a cross-SQL flow-style query language for functional data modeling and interactive data analysis.
Try Wvlet! Documentation Star 37

Wvlet is a cross-SQL flow-style query language for functional data modeling and interactive data analysis.

Try Wvlet! Documentation Star 37

```
| -- Apply a filtering condition  
| where _quantity > 10.0  
| -- Apply aggregation by columns  
| group by _returnflag, _linestatus  
| agg  
|   _quantity.sum as qty,  
|   _extendedprice.sum as price,  
|   _discount.sum as discount_avg,  
|   order by _returnflag, _linestatus, qty desc  
| exclude discount  
debug (line:6) group by _returnflag, _linestatus
```

_returnflag	_linestatus	arbitrary(_orderkey)	arbitrary(_partkey)	arbitrary(_l_supp
N	F		197	188
N	O		1	3552
A	F		3	1285
R	F		3	43

4 rows

```
wv -- Scan from a file  
from 'linelitem_pqquet'  
-- Apply a filtering condition  
where _quantity > 10.0  
-- Aggregate by columns  
group by _returnflag, _linestatus  
-- Compute aggregate  
agg  
| _quantity.sum as qty,  
| _extendedprice.sum as price,  
| _discount.sum as discount_avg,  
| order by _returnflag, _linestatus, qty desc  
| exclude discount
```



Appendix

Wvlet Internals: Standard Library

- Wvlet defines standard data types and functions, which define how to generate SQL
 - example: `x.to_int.round(1)` => `cast(x as bigint).round(1)` => `round(cast(x as bigint),1)`

```
package wvlet.standard

type int:
    def to_int: int = sql"cast(${this} as bigint)"
    def to_long: long = sql"cast(${this} as bigint)"
    def to_float: float = sql"cast(${this} as double)"
    def to_double: double = sql"cast(${this} as double)"
    def to_boolean: boolean = sql"cast(${this} as boolean)"
    def to_string: string = sql"cast(${this} as varchar)"

    def or_else(other:int): int = sql"coalesce(${this},${other})"

    def round(decimal:int=0): double = sql"round(${this}, ${decimal})"

    def in(v:any*): boolean = sql"${this} in (${v})"
    def not_in(v:any*): boolean = sql"${this} not in (${v})"

    def between(l:int, r:int): boolean = sql"${this} between ${l} and ${r}"
end
```

Wvlet Internals: DBMS Specific Functions

- Function can have different implementations depending on the target databases (Trino/Hive/DuckDB, etc.)
 - `x.count_approx_distinct`
 - Trino: `approx_distinct(x)`
 - DuckDB: `approx_count_distinct(x)`
- Works for consuming the differences between SQL dialects and UDFs

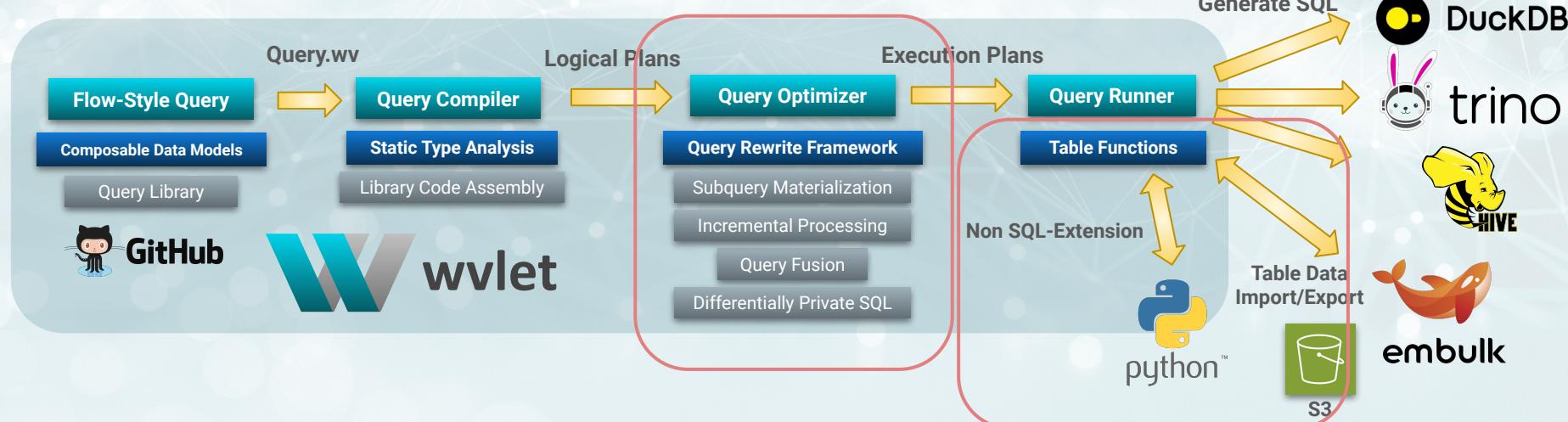
```
-- An array created by 'group by' clause
type array[A]:
    def length: int = sql"length(${this})"
    def size: int = sql"length(${this})"
    def get(index: int): A = sql"${this}[$index]"
    def count: int = sql"count(*)"
    def count_distinct: int = sql"count(distinct ${this})"
    def count_if(cond: boolean): int = sql"count_if(${cond})"

    -- Fast and memory-efficient approximate counting of distinct elements
    def count_approx_distinct in trino: int = sql"approx_distinct(${this})"
    def count_approx_distinct in duckdb: int = sql"approx_count_distinct(${this})"
```



Extending Wvlet

- **Compiler plugins (to be designed #185)**
 - Add a custom rule set to optimize logical plans and execution plans.
 - Optimization rules: [Query fusion](#) optimization, [subquery materialization](#), [incremental processing](#), etc.
 - Security rules (e.g., forbidding local file access operators at the cloud environment)
- **Table functions receive table-value data and output table value data**
 - Not limited to SQL, we will be able to invoke ML algorithms, calling Embulk, or issuing SQL queries to different query engines by generating such execution plans.



Writing A Compiler From Scratch? Leverage LLMs

- Developing a compiler is a challenge with known difficulties and known solutions
 - = solved problem
- By leveraging LLM-based tools like GitHub Copilot, we can accelerate the development
- LLMs excel at utilizing well-known solutions from open-source code.

The screenshot shows a code editor window with a dark theme. At the top right, a teal bar says "Indicate what you want to code". Below the code, another teal bar says "GitHub Copilot suggests code in a second". The code itself is a Scala implementation of a parser for different types of database joins. It uses pattern matching on tokens to determine the type of join and then performs specific actions based on that type. The code includes imports for `EmptyRelation` and `JoinCriteria`, and uses methods like `relationPrimary` and `joinCriteria`.

```
def parseJoin: Join =  
  val t = scanner.lookAhead()  
  t.token match  
    case WvletToken.CROSS =>  
      consume(WvletToken.CROSS)  
      consume(WvletToken.JOIN)  
      val right = relationPrimary()  
      Join(JoinType.CrossJoin, EmptyRelation(t.span), right, NoJoinCriteria, spanFrom(t))  
    case WvletToken.JOIN =>  
      consume(WvletToken.JOIN)  
      val right = relationPrimary()  
      val joinOn = joinCriteria()  
      Join(JoinType.InnerJoin, EmptyRelation(t.span), right, joinOn, spanFrom(t))  
    case WvletToken.LEFT | WvletToken.RIGHT | WvletToken.INNER | WvletToken.FULL =>  
      val joinType =  
        t.token match  
          case WvletToken.LEFT =>  
            JoinType.LeftOuterJoin  
          case WvletToken.RIGHT =>  
            JoinType.RightOuterJoin  
          case WvletToken.INNER =>  
            JoinType.InnerJoin  
          case WvletToken.FULL =>  
            JoinType.FullOuterJoin  
          case _ =>  
            unexpected(t)  
      consume(t.token)  
      consume(WvletToken.JOIN)  
      val right = relationPrimary()  
      val joinOn = joinCriteria()  
      Join(joinType, EmptyRelation(t.span), right, joinOn, spanFrom(t))  
    case _ =>  
      unexpected(t)
```

Wvlet is easy to learn, even for Large Language Models (LLMs)

- An LLM Agent, created from Wvlet documentation, can successfully convert TPC-H SQL queries into Wvlet syntax
 - Wvlet: Query Syntax documentation ([link](#))

Query Syntax

References

- Quick Start
- Relational Operators
- Expressions
- Metadata Functions
- Data Models

Introduction

Wvlet is a query language designed to be more human-readable and easier to write than SQL. If you already familiar to SQL, you will find it's easy to learn the syntax of wvlet as there are a lot of similarities between wvlet and SQL. Even if you are new to SQL, no worries! You can start learning wvlet from scratch. If you know about DataFrame in Python, it will help you understand the wvlet query language as chaining relational operators in the flow-style is quite similar to using DataFrame API.

Wvlet queries start with `from` keyword, and you can chain multiple relational operators to process the input data and generate the output data. The following is a typical flow of chaining operators in a wvlet query:

```
from ...           -- Scan the root data
    filter ...      -- [optional] Apply more filtering conditions
    where ...        -- Add filtering conditions
    add ...          -- Transform a subset of columns
    transform ...    -- Transform the columns
    agg ...          -- Add group aggregation expressions, e.g., _count, _sum
    where ...        -- Apply filtering conditions for groups (e.g., HAVING clause in SQL)
    exclude ...
    except ...
    shift ...
    select ...
    shift ...
    sort ...
    order by ...
    limit ...        -- Limit the number of rows to output
```

Unlike SQL, whose queries always must follow the `SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ... LIMIT ...` structure, wvlet uses the [flow-style syntax](#) to match the syntax order with the data processing order as much as possible to facilitate more intuitive query writing.

Some operators like `add`, `transform`, `agg`, `exclude`, `shift`, etc. are not available in the standard SQL, but these new operators have been added for reducing the amount of code and making the query more readable and easier to compose. Eventually, these operators will be translated into the equivalent SQL syntax.

You
Write some TPC-H queries in Wvlet

Agent

Certainly! I'll provide you with some examples of TPC-H queries transformed into oriented ad-hoc queries. Here are a few examples:

1. Query 1 - Pricing Summary Report:

```
from lineitem
where l_shipdate <= date('1998-12-01') - interval '90' day
group by
    l_returnflag,
    l_linenstatus
agg
    sum_qty = _.l_quantity.sum,
    sum_base_price = _.l_extendedprice.sum,
    sum_disc_price = (_.l_extendedprice * (1 - _.l_discount)).sum,
    sum_charge = (_.l_extendedprice * (1 - _.l_discount) * (1 + _.l_tax)).sum,
    avg_qty = _.l_quantity.ave,
    avg_price = _.l_extendedprice.ave,
    avg_disc = _.l_discount.ave,
    count_order = _.count
order by
    l_returnflag,
    l_linenstatus
```

Original TPC-H Q1 SQL



TREASURE DATA

TPC-H Q1 in Wvlet

wv: Interactive Query Editor

- Available with [brew install wvlet/wvlet/wvlet](#)
 - Supports Trino, DuckDB via profile settings
- Shortcut keys for checking intermediate schema and results
 - **ctrl-j, ctrl-d** (describe the schema at the line)
 - **ctrl-j, ctrl-t** (test run the subquery upto the line)
 - **ctrl-j, ctrl-r** (run the whole query)

```
wv> from cost_components
| where time.within('-3M/now')
| where aws_account_owner != 'security'
| add service_name = coalesce(user_service, cost_category)
| -- type ctrl+j ctrl+t (test run) to debug the intermediate query result
| add td_time_string(time, 'M!') as month
| pivot on month in ('2024-06', '2024-07', '2024-08')
| group by service_name
| agg cost.sum.round(1)
| order by `2024-08` desc
| limit 10;
debug (line:6): add td_time_string(time, 'M!') as month
```

cost_category string	aws_account_code string	aws_account_owner string	user_service string
Production	td	engineering	waf-event-collector3
Production	kitchen	engineering-lead	core-eks-app-ronald-testing
Non Production	td	engineering	waf-event-collector3
Production	kitchen	engineering-lead	core-eks-app-scp-foo
Non Production	td	engineering	waf-event-collector3
Production	kitchen	engineering-lead	core-eks-app-scp-foo
Non Production	td	engineering	waf-event-collector3
Production	kitchen	engineering-lead	core-eks-app-scp-foo
Production	td	engineering	waf-event-collector3
Non Production	td	engineering	waf-event-collector3

Subquery Result

```
describe (line:4): where aws_account_owner != 'security'
```

column_name string	column_type string
cost_category	string
aws_account_code	string
aws_account_owner	string
user_service	string
service_owner_team	string
user_site	string
user_cluster	string
customer_id	string
plan_type	string
resource_product_region	string
resource_product_code	string
resource_usage	string
net_effective_cost	double
effective_cost	double
public_cost	double
cost	double
user_stage	string
time	long

Intermediate Table Schema

```
wv> from cost_components
| where time.within('-3M/now')
| -- type ctrl+j ctrl+d (describe) here to show the intermediate schema
| where aws_account_owner != 'security'
| add service_name = coalesce(user_service, cost_category)
| add td_time_string(time, 'M!') as month
| pivot on month in ('2024-06', '2024-07', '2024-08')
| group by service_name
| agg cost.sum.round(1)
| order by `2024-08` desc
| limit 10;
```



Design Philosophy of Wvlet

- Link: <https://wvlet.org/wvlet/docs/internal/design>
- Matching syntax order with data flow
 - Typing from left-to-right, top-to-bottom order
- Use only lowercase keywords
 - Forbid mixing SELECT and select
- One operation at a time
 - SQL's SELECT operator is too powerful
 - Breaking down SELECT statement into simple operators (add, transform, exclude, shift, etc)
- Human and machine friendly-syntax
 - Query lines form subqueries
 - Trailing comma support is essential to correctly extract subqueries in an executable form
 - Enable in-query data/schema inspection

```
SELECT
    sum(c1),
    -- Rename c2 with an alias
    c2 as c2_new,
    -- skip c3 for exclusion
    -- Add a new computed column
    c4 + c5 as c101,
    -- Shift c6 and c7 to the end
    c8,
    ...
    ...,
    c100,
    c6,
    c7,
    FROM tbl
```

SQL

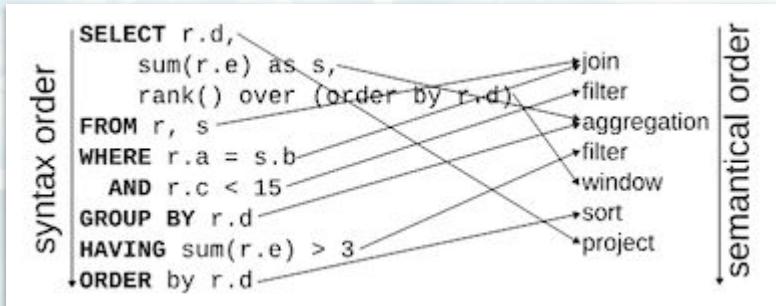


```
wv> from tbl
      -- Add a simple aggregation
      add c1.sum
      -- Rename c2 with an alias
      transform c2 as c2_new
      -- Remove c3 from the result
      exclude c3
      -- Add a new computed column
      add c4 + c5 as c101
      -- Shift c6 and c7 to the end
      shift to right c6, c7
```

Wvlet

Related Work: SaneQL (Thomas Neumann, et al. 2024)

- The syntactic order of SQL doesn't match with the semantic order of data processing
- Redesigned SQL by using a function chain syntax
 - [A Critique of Modern SQL And A Proposal Towards A Simple and Expressive Query Language \(CIDR 24\)](#)



```
let min_cost_for_part(p_partkey) :=
    partsupp.filter(ps_partkey=p_partkey)
    .join(supplier, s_suppkey=ps_suppkey)
    .join(nation, s_nationkey=n_nationkey)
    .join(region.filter(r_name='EUROPE'),
          n_regionkey=r_regionkey)
    .aggregate(min(ps_supplycost)),
part
    .filter(p_size = 15 && p_type.like('%BRASS'))
    .join(partsupp, p_partkey = ps_partkey)
    .join(supplier, s_suppkey = ps_suppkey)
    .join(nation, s_nationkey = n_nationkey)
    .join(region.filter(r_name='EUROPE'),
          n_regionkey=r_regionkey)
    .filter(ps_supplycost=min_cost_for_part(p_partkey)) ...
```



Related Work: PRQL (Prequel)

- Pipelined Relational Query Language (Since 2022) <https://prql-lang.org/>
- Written in Rust, compiling PRQL to SQL
 - Syntax is a bit far from SQL, more like DataFrame API or programming language syntax

PRQL

```
from tracks
filter artist == "Bob Marley" # Each line transforms the previous result
aggregate {
    plays = sum plays,
    longest = max length,
    shortest = min length,      # Trailing commas are allowed
}
```

SQL

```
SELECT
    COALESCE(SUM(plays), 0) AS plays,
    MAX(length) AS longest,
    MIN(length) AS shortest
FROM
    tracks
WHERE
    artist = 'Bob Marley'
```

