Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: jscit.nit.ac.ir
Volume 11, Number 4, Winter 2023, pp. 19-35
Received: 08/26/2022, Revised: 06/06/2022, Accepted: 01/03/2023

Automatic NoSQL Schema Design: A Workload-Driven Schema Design Approach for NoSQL Wide Column Stores

Maryam Mozaffari[1*] and Eslam Nazemi[2]

1*- Department of Computer Engineering, Gonbad Kavoos Branch, Islamic Azad University, Gonbad Kavoos, Iran
2- Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran

[1*] m.mozaffari@gonbadiau.ac.ir, [2]nazemi@sbu.ac.ir

Corresponding author's address: Maryam Mozaffari, Department of Computer Engineering, Gonbad Kavoos Branch, Islamic Azad University, Gonbad Kavoos, Iran

Abstract- NoSQL systems are suitable solutions for big data projects and offer a high level of flexibility in design. A good and efficient schema design for NoSQL wide column stores is not only based on the application's conceptual data model but also on the queries defined in an application's workload. In these databases, a manual schema design relies on rules of thumb to choose a good schema. Utility these rules without practices is a big challenge in this area. Because these rules are vague and generic, and must be adapted to each application. One of the ways that researchers use to overcome this challenge is presenting automated schema design. The main contribution of this paper is automated schema design for NoSQL wide column stores. This research proposes a workload-driven approach for the mapping from the application conceptual data model to a database schema with a goal of optimizing query performance. This approach uses workload information to achieve a good workload performance, which result in an optimized schema by minimizing the number of requests to the wide column stores. The experimental results show that automated schema generated by the proposed approach leads to a good workload performance.

Keywords- NoSQL wide column stores, Automated schema design, Self-tuning, Schema optimization, Workload performance.

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
جلد ۱۱، شماره ۴، زمستان ۱۴۰۱ ، صفحه ۱۹-۳۵
دریافت: ۱۴۰۱/۰۶/۰۴، بازنگری: ۱۴۰۱/۰۳/۱۶، پذیرش: ۱۴۰۱/۱۰/۱۳

طراحی شِما خودکار NoSQL: یک روش طراحی شِما مبتنی بر بارکاری برای پایگاه داده ستون‌گسترده NoSQL

مریم مظفری*[۱]، اسلام ناظمی[۲]
*۱- گروه مهندسی کامپیوتر، واحد گنبدکاووس، دانشگاه آزاد اسلامی، گنبدکاووس، ایران.
۲- دانشکده مهندسی و علوم کامپیوتر، دانشگاه شهید بهشتی، تهران، ایران.
[1*]m.mozaffari@gonbadiau.ac.ir, [2]nazemi@sbu.ac.ir

* نشانی نویسنده مسئول: مریم مظفری، گنبدکاووس، دانشگاه آزاد اسلامی واحد گنبدکاووس، گروه مهندسی کامپیوتر

چکیده- سیستم‌های پایگاه داده NoSQL راهکارهای مناسبی برای پروژهای کلان داده هستند و از انعطاف‌پذیری بالایی در طراحی برخوردارند. در پایگاه داده ستون‌گسترده NoSQL، طراحی یک شِما کارامد و مناسب نه تنها وابسته به مدل داده مفهومی، بلکه به پرس‌وجوهای برنامه هم وابسته است. در این پایگاه داده‌ها ، طراحی شِما دستی وابسته به قوانین سرانگشتی برای انتخاب شِمای مناسب است. بکارگیری این قوانین بدون تجربه‌های عملی یک چالش بزرگ در این زمینه است. چرا که این قوانین مبهم و متناقض هستند. یکی از راه‌هایی که پژوهشگران برای غلبه بر چالش مذکور پیش گرفته‌اند، ارائه روش‌های خودکارسازی طراحی شِمای پایگاه داده است. هدف اصلی این تحقیق خودکارسازی طراحی شِمای پایگاه داده ستون-گسترده NoSQL است که بدین منظور یک روش مبتنی بر بارکاری برای نگاشت مدل داده مفهومی به شِما پایگاه داده ستون گسترده با هدف بهینه-سازی کارایی بارکاری ارائه می‌شود. در این روش از اطلاعات بارکاری برای رسیدن به کارایی بهتر پرس‌وجوها استفاده شده است که با کمینه کردن تعداد درخواست‌ها به پایگاه داده منجر به طراحی شِمای بهینه‌یافته می‌شود. نتایج حاصل شده از آزمایش‌ها نشان می‌هد که شِما خودکار تولید شده از روش پیشنهادی منجر به کارایی خوب بارکاری می‌گردد.

واژه‌های کلیدی: پایگاه داده ستون گسترده NoSQL، طراحی شِما خودکار، خود-تنظیمی، بهینه‌سازی شِما، کارایی بارکاری

۱- مقدمه

در سال‌های اخیر، پیچیدگی و گستردگی سیستم‌های مدیریت پایگاه داده به حدی رسیده است که کنترل و نگهداری آنها بسیار پرهزینه و زمانبر است و در بعضی از موارد از عهده نیروی انسانی خارج است. به دلیل پیچیدگی روزافزون این سیستم‌ها، مدیران پایگاه داده با چالش‌های بیشتری روبرو هستند که مدیریت این سیستم‌ها را دشوار می‌نماید. به همین علت مطالعات بسیاری در مورد اضافه نمودن قابلیت خودتطبیقی[۱] به این سیستم‌ها انجام شده است که حاصل آن طراحی و توسعه سیستم‌های مدیریت پایگاه داده خودمختار[۲] است. قابلیت مذکور، سیستم پایگاه داده را قادر به مدیریت و نگهداری از خود بدون دخالت انسان می‌سازد.

ویژگی خودتنظیمی[۳] یکی از ویژگی‌های پایگاه داده خودمختار است که اشاره به تنظیم خودکار شِمای پایگاه داده[۴] دارد. این ویژگی در پایگاه داده رابطه‌ای به خودکارسازی طراحی شِمای فیزیکی پایگاه داده می‌پردازد. پایگاه داده رابطه‌ای دارای مدل-های داده ثابت (شِماهای ثابت و از پیش تعریف شده) و همچنین رویه‌های خوش تعریف و استاندارد برای تبدیل مدل داده مفهومی پایگاه‌داده به شِمای منطقی نرمال شده است. در این پایگاه داده شِمای منطقی تعیین کننده شِمای فیزیکی متشکل از جداول پایه می‌باشد. در شِمای فیزیکی این جداول می‌توانند بهینه شوند و ساختارهای فیزیکی نظیر اندیس‌ها و دیدهای ذخیره شده به آنها اضافه شود. بنابراین پایگاه داده رابطه‌ای تمایز شفافی بین شِمای منطقی و فیزیکی قایل است و

تنظیم طراحی فیزیکی شامل افزودن ساختارهای فیزیکی به مجموعه‌ای از جداول ثابت برای بارکاری مشخص است. بنابراین در این پایگاه داده‌ها، مساله طراحی فیزیکی خودکار پایگاه داده[۵]، یافتن یک پیکربندی مناسب (مجموعه‌ای از ساختارهای فیزیکی) برای یک بارکاری[۶] مشخص(W) و بودجه ذخیره‌سازی مشخص (B) است که منجر به کمترین هزینه اجرای پرس‌وجوهای موجود در بارکاری می‌شود [۱]. محققان ابزارهای بسیاری را در این زمینه ارائه کرده‌اند [۱۰-۲].

در مقابل، در پایگاه داده غیر رابطه‌ای NoSQL طراحی مدل‌های داده بسیار انعطاف پذیر یا بدون شِما[۷] هستند. انعطاف پذیری بالا این مدل‌های داده منجر به ساختار داده‌های با آزادی بیشتر از پایگاه داده رابطه‌ای می‌شود. از این‌رو مساله طراحی شِما برای پایگاه داده غیررابطه‌ای از طراحی شِمای فیزیکی رابطه‌ای متفاوت است. طراحی شِما در این پایگاه داده‌ها فقط محدود به ساختارهای فیزیکی نمی‌شود و از آنجاییکه مدل داده منطقی هم بر مبنای پرس‌وجوهای برنامه قابل تغییر است نیز در بر می‌گیرد. از این‌رو، طراحی یک شِما کارامد و مناسب نه تنها فقط وابسته به مدل داده مفهومی، بلکه به پرس‌وجوهای برنامه هم وابسته است.

سیستم پایگاه داده NoSQL از بسیاری از مفاهیم موجود در پایگاه داده رابطه‌ای نظیر عملیات پیوند جداول پشتیبانی نمی‌کند. زیرا با توجه به توزیع داده ها در چندین ماشین، عملیات پیوند بسیار پرهزینه است. در این پایگاه داده‌ها برای طراحی شِمای بهینه پایگاه داده به دنبال استفاده از فنون غیرنرمال سازی هستند تا بتوان به پرس‌وجوهای پیچیده فقط با یک درخواست به پایگاه داده پاسخ داده شود. بنابراین روش‌های ارائه شده در زمینه طراحی و تنظیم شِمای پایگاه داده NoSQL، از فنون غیرنرمال سازی پایگاه داده برای جلوگیری از عملیات پیوند استفاده نموده‌اند و به دنبال طراحی شِمای بهینه‌ای از پایگاه داده برای بالا بردن کارایی پرس‌وجوها هستند.

در پایگاه داده ستون‌گسترده[۸] که نوع خاصی از پایگاه داده غیررابطه‌ای NoSQL است، جداولی از رکوردها ایجاد می‌شود که هر رکورد دارای یک کلید است و مجموعه ستون‌های هر رکورد، از پیش تعریف شده نیستند. همچنین هر رکورد می‌تواند دارای ستون‌های مختلفی نیز باشد. این جداول در پایگاه داده ستون‌گسترده، گروه ستون[۹] (CF) نامیده می‌شوند [۱۱]. بنابراین مساله اصلی طراحی شِما در این نوع از پایگاه داده NoSQL، تعیین جداول و اطلاعاتی است که در هر جدول برای یک بارکاری مشخص باید ذخیره شود تا هزینه اجرای پرس‌وجوهای بارکاری کمینه گردد. از نمونه‌های این نوع پایگاه داده می‌توان به کاساندرا و اچ‌بیس[۱۰] اشاره کرد.

در این تحقیق یک روش طراحی شِما خودکار برای پایگاه داده ستون‌گسترده ارائه شده است که به بهینه‌سازی شِمای پایگاه داده برای کل بارکاری برنامه یعنی پرس‌وجوهای خواندن و نوشتن می‌پردازد. روش پیشنهادی شامل فرایند نگاشت مدل داده مفهومی به شِما پایگاه داده ستون‌گسترده کاساندرا است که از اطلاعات بارکاری برای رسیدن به کارایی بهتر پرس‌وجوها استفاده شده است. این روش با کمینه کردن تعداد درخواست‌ها به پایگاه داده منجر به طراحی شِمای بهینه‌یافته می‌شود.

در ادامه، مقاله بدین‌صورت سازماندهی شده است: در بخش ۲ مفاهیم پایگاه داده ستون‌گسترده با تمرکزی بر مدل داده کاساندرا ارائه می‌شود. در بخش ۳ مهمترین کارهای مرتبط با موضوع این پژوهش بررسی خواهند شد. بخش ۴ شامل روش پیشنهادی طراحی شِما خودکار در پایگاه داده ستون‌گرا NoSQL است. در بخش ۵ نتایج حاصل از روش پیشنهادی تشریح می‌شود. در پایان نتیجه‌گیری مختصری در خصوص پژوهش و کارهای آینده در بخش ۶ معرفی شده‌اند.

۲- پایگاه داده ستون‌گسترده

پایگاه داده ستون گسترده برای ذخیره سازی جداول داده به صورت ستون‌هایی از داده، به جای سطرهایی از داده طراحی شده است. در حالیکه این پایگاه داده مدلی شبیه به پایگاه داده سنتی رابطه‌ای دارد ولی از کارایی و مقیاس پذیری بالایی برخوردار است. تمرکز اصلی این تحقیق در روی پایگاه داده ستون‌گسترده کاساندرا است، از این‌رو در ادامه به توصیف ویژگی‌ها و ساختار داده این پایگاه داده پرداخته می‌شود.

آپاچی کاساندرا یک سیستم پایگاه داده توزیع شده و متن باز است که برای مدیریت داده‌های بزرگ طراحی شده است. از خصوصیات کلیدی پایگاه‌داده کاساندرا می‌توان به قابلیت تحمل خطا[۱۱]، مقیاس‌پذیری خطی[۱۲] و دسترس‌پذیری بالا[۱۳] اشاره کرد. این پایگاه‌داده از ابتدا به گونه‌ای طراحی شده است که به شدت مقیاس‌پذیر بوده و به سادگی می‌تواند روی سرورهایی که در مراکز داده[۱۴] مختلف پراکنده شده‌اند، توزیع شود [۱۲]. بزرگ‌ترین مفهوم داده‌ای در کاساندرا Keyspace است و معادل مفهوم پایگاه داده در مدل رابطه‌ای است. مفهوم داده‌ای بعدی، گروه ستون است که نگه‌دارنده مجموعه از سطرهای داده‌ای است و هر کدام از سطرها، حاوی مجموعه از ستون‌های داده‌ای با ساختار

مشابه هستند. در حقیقت درمدل داده کاساندرا جداولی از رکوردها ایجاد می‌شود که هر رکورد دارای یک کلید اصلی است و مجموعه ستون‌های هر رکورد، از پیش تعریف شده نیستند. همچنین هر رکورد می‌تواند دارای ستون‌های مختلفی باشد و درون ستون‌ها داده‌ها به صورت کلید/مقدار ذخیره می‌شوند. این جداول در پایگاه داده ستون گسترده، گروه ستون (CF) نامیده می‌شوند. یک گروه ستون (CF) با علامت سه بخشی زیر نمایش داده می‌شود:

[Row Key][Column Key][Column Values]

در نسخه اولیه از کاساندرا، رابط کاربری برنامه-نویسی Thrift است که گروه ستون را به عنوان ساختاری برای ذخیره و سازماندهی داده‌ها معرفی می‌کند. در نسخه-های جدیدتر، رابط CQL[۱۵] معرفی شده است که از واژه جدول بجای واژه گروه ستون نام می‌برد. بنابراین این دو واژه در کاساندرا با هم معادل هستند و جدول یک دید دوبعدی از گروه ستون چند بعدی است.

یک گروه ستون مجموعه‌ای از ردیف‌ها است که هر ردیف شامل ستون‌هایی با ساختار مشابه است و دارای یک کلید یکتا است که این کلید مشخص می‌کند هر ردیف در روی کدام گره در کلاستر ذخیره گردد. از این رو عنصر اول در علامت سه‌بخشی مذکور اشاره به کلید ردیف دارد. هر ستون در یک ردیف به طور اختیاری دارای کلید ستون یکتایی هستند که مقدار هر ستون در یک ردیف را به صورت منحصر به فرد شناسایی می‌کند. در هر ردیف، ستون‌ها با استفاده از کلید ستون مرتب می‌شوند. بنابراین عنصر دوم در علامت سه‌بخشی، کلید ستون است. یک گروه ستون بدون کلید ستون می‌تواند دارای ردیف‌هایی فقط با یک ستون باشد. هر دو کلید می‌توانند ساده (تک‌ستونی) یا مرکب (چندستونی) باشد. ترکیب کلید ردیف و کلید ستون، کلید اصلی نام دارد. عنصر سوم مقادیر ستون است که یک مقدار یا مجموعه‌ای از مقادیر را ذخیره می‌کند.

به طور مشابه معادل یک گروه ستون، یک جدول CQL است که مجموعه‌ای از افرازها است و با علامت سه بخشی زیر نمایش داده می‌شود:

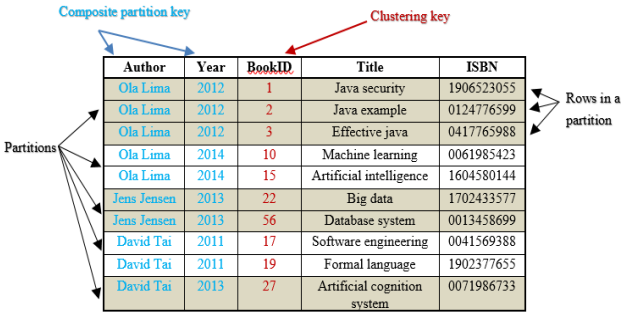
[Partition Key] [Clustering Key] [Non-key Columns]

هر افراز شامل ردیف هایی با ساختار مشابه است و دارای یک کلید یکتا به نام کلید افراز است. این کلید مشخص می‌کند که هر افراز در روی کدام گره ذخیره گردد. در حقیقت ردیف‌ها را در روی گره‌های مختلف افراز می‌کند. هر ردیف در یک افراز به طور اختیاری دارای کلید خوشه-بندی یکتایی است که مقدار هر ردیف در افراز را به صورت منحصر به فرد شناسایی می‌کند. در هر افراز، ردیف‌ها با استفاده از کلید خوشه‌بندی مرتب می‌شوند. ترکیب کلید افراز و کلید خوشه‌بندی، کلید اصلی نام دارد که یک ردیف در جدول را به صورت یکتا شناسایی می‌کند. یک جدول بدون کلید خوشه‌بندی می‌تواند دارای افرازهایی فقط با یک ردیف باشد. ولی جدولی با کلید خوشه‌بندی می‌تواند دارای افرازهای چند-ردیف باشد. شکل ۱ نمای جدولی با افراز تک-ردیف را نمایش می‌دهد.

کاساندرا یک پایگاه داده مبتنی بر درخواست است. بدین معنی که بر اساس پرس‌وجوهای که بر روی داده‌های در آینده اجرا خواهند شد، می‌بایست داده‌ها ذخیره شوند. از طرفی نحوه بازیابی داده‌ها نیز کاملا وابسته به کلید اصلی جداول است و روی فیلدهای غیرکلید نمی‌توان جستجو انجام داد. کاساندرا از زبان پرس‌وجو CQL که بسیار شبیه SQL است، استفاده می‌کند. زبان CQL برخلاف SQL از عملیاتی نظیر پیوند و زیرپرس‌وجو پشتیبانی نمی‌کند. کاساندرا به این پرس‌وجوهای پیچیده از طریق روش‌هایی نظیر غیرنرمال‌سازی و لایه کاربرد پاسخ می-دهد. در حقیقت اگر پرس‌وجوی خواندن فقط شامل یک موجودیت در مدل مفهومی است، این موجودیت تبدیل به یک جدول می‌شود و ویژگی‌های آن به ستون‌های جدول نگاشت می‌شوند. ولی برای اجرای پرس‌وجوهای خواندن پیچیده که دارای عملگر پیوند هستند دو امکان مختلف در کاساندرا وجود دارد:

غیرنرمالسازی: برای پرس‌وجوهایی که شامل عملگر پیوند هستند با استفاده از روش غیرنرمالسازی یک جدول ایجاد می‌شود که تمامی داده‌های مربوط به پرس‌وجو مذکور در این جدول جمع-آوری می‌شود. بنابرای در این روش، به هر پرس‌وجوی خواندن فقط توسط یک درخواست به پایگاه داده پاسخ داده می‌شود(با ارسال یک درخواست به جدول ایجاد شده).

لایه کاربرد (پیوند سمت کلاینت[۱۶]): در این روش عملیات پیوند در لایه کاربرد انجام می‌شوند. بدین صورت که چندین پرس-وجوی select برای واکشی داده‌ها از جداول پیوند ایجاد می‌شود. سپس نتایج جزئی پرس‌وجوهای select برای پاسخ به پرس-وجوی پیوند با هم ترکیب می‌شوند.



Author	Year	BookID	Title	ISBN
Ola Lima	2012	1	Java security	1906523055
Ola Lima	2012	2	Java example	0124776599
Ola Lima	2012	3	Effective java	0417765988
Ola Lima	2014	10	Machine learning	0061985423
Ola Lima	2014	15	Artificial intelligence	1604580144
Jens Jensen	2013	22	Big data	1702433577
Jens Jensen	2013	56	Database system	0013458699
David Tai	2011	17	Software engineering	0041569388
David Tai	2011	19	Formal language	1902377655
David Tai	2013	27	Artificial cognition system	0071986733

شکل ۱: نمای جدولی با افراز تک-ردیف

۳- کارهای مرتبط

در زمینه تنظیم خودکار طراحی فیزیکی برای پایگاه داده رابطه-ای تحقیقات زیادی انجام شده است و محققان ابزارهای راهنمایی[۱۷] نظیر راهنمای طراحی IBM DB2 [۸،۹]، راهنمای تنظیم SQL در اوراکل [۴] و راهنمای تنظیم پایگاه داده Microsoft SQL Server [۲] ارائه کرده‌اند که در محصولات تجاری پایگاه داده پیاده‌سازی شده‌اند. این راهنماها برون‌خط هستند و فقط به دنبال یافتن مجموعه بهینه‌ای از ساختارهای فیزیکی نظیر اندیس‌ها و دیدهای ذخیره شده برای بارکاری برنامه هستند. همچنین محققان به ارائه روش‌های برخط، نظیر [۳،۶،۷]، در این زمینه پرداخته‌اند که قادر به نظارت پیوسته بر بارکاری و وضعیت سیستم و تنظیم پیکربندی ساختارهای فیزیکی پایگاه داده با تغییرات بوجود آمده در آنها هستند.

در [۱۰] یک ابزار طراحی فیزیکی خودکار برای سیستم پایگاه داده رابطه‌ای ستون‌گرا C-store [۱۳] ارائه شده است. این ابزار مجموعه بهینه‌ای از دیدهای ذخیره شده و اندیس‌های خوشه‌ای متناظر را ایجاد می‌کند.

محققان در [۱۴] اولین سیستم مدیریت پایگاه داده خود-گرداننده[۱۸] بنام Peloton را توسعه داده‌اند. سیستم مدیریت پایگاه داده خود-گرداننده می‌تواند خودش را بدون هیچ گونه دخالت انسان پیکربندی، تنظیم و بهینه کند. Peloton یک سیستم مدیریت پایگاه داده رابطه‌ای است که برای عملیات خودمختار طراحی شده و قادر به کنترل تمام جنبه‌های سیستم پایگاه داده توسط یک مولفه طرح‌ریزی یکپارچه است.

در زمینه خودکارسازی و طراحی شِمای بهینه پایگاه داده NoSQL تحقیقات کمی انجام شده است [۱۵-۲۲]. روش‌های ارائه شده در این زمینه اغلب وابسته به فنون غیرنرمال سازی پایگاه داده برای بهینه‌سازی شِما و بهبود کارایی پرس‌وجوها بارکاری هستند. در ادامه به بررسی هر یک از این تحقیقات می-پردازیم.

محققان در [۱۵]، یک روش سیستماتیک برای طراحی خودکار شِمای پایگاه داده ستون‌گرا NoSQL ارائه نموده‌اند. هدف اصلی این روش بهینه‌سازی شِمای پایگاه داده برای پرس‌وجوهای خواندنی است. بدین صورت که هر پرس‌وجوی خواندن تنها با یک درخواست به پایگاه داده اجرا شود. این روش شامل دو مرحله ایجاد شِمای خودکار از مدل مفهومی اولیه و ارزیابی شِما است. در مرحله ایجاد شِما، ابتدا با تحلیل پرس‌وجوهای انتخاب و پیوند و روش‌های غیرنرمال‌سازی تمام شِماهای ممکن شناسایی می‌شود. این شِماها شامل جداول ستون‌گرایی هستند که منجر به اجرای پرس‌وجوها، تنها با یک درخواست به پایگاه داده می-شوند. سپس با حذف یا ادغام جداول در شِماها بر طبق شرط-های تعریف شده، این شِماها بهینه‌سازی می‌شوند. در مرحله ارزیابی شِما با استفاده از مجموعه‌ای از متریک‌ها و یک تابع امتیازدهی، بهترین شِما از بین شِماهای پیشنهادی انتخاب می-شود.

روش‌های ارائه شده در [۱۶] و [۱۷] به طراحی شِما پایگاه داده برای بهبود عملکرد پرس و جوهای خواندن با کاهش تعداد دسترسی به پایگاه‌های داده NoSQL پرداخته‌اند. این روش‌ها بر مبنای قوانین نگاشت، یک مدل داده مفهومی را به یک مدل داده NoSQL تبدیل می‌کنند که از اطلاعات بارکاری برای انتخاب یک شِما بهینه بهره می‌برند. در [۱۶] یک ابزار مدل‌سازی داده بنام KMD برای طراحی شِمای خودکار پایگاه داده ستون-گسترده کاساندرا بر طبق متدلوژی مدلسازی مبتنی بر پرس‌وجو ارائه شده است. این متدلوژی با استفاده از تحلیل پرس‌وجوها و قوانین و الگوهای نگاشت مشخص، شِمای مفهومی را به شِمای منطقی تبدیل می‌کند. این روش منجر به بهترین کارایی پرس-وجوی خواندن می‌شود. به طوریکه هر پرس‌وجوی خواندن پیچیده تنها با یک درخواست به پایگاه داده پاسخ داده می‌شود. روش ارائه شده در [۱۷] شامل فرایند تبدیل شِمای مفهومی به شِمای منطقی در پایگاه داده سندگرا NoSQL و پیاده‌سازی فیزیکی آن است. بر طبق این روش، برای جمع‌آوری و تحلیل اطلاعات بارکاری، طبق قانون ۸۰-۲۰، بر روی ۲۰ درصد از عملیات پرتکرار بارکاری تمرکز می‌شود. در این روش تحلیل بارکاری شامل شناسایی موجودیت‌ها و ارتباط‌هایی در مدل مفهومی است که توسط پرس‌وجوهای بارکاری مکررا استفاده شده‌اند. همانطور که گفته شد، این روش توسط کاهش تعداد دسترسی به پایگاه داده، کارایی پرس‌وجوها در روی مستندات NoSQL را بهبود می‌بخشد.

در [۱۸] ابزاری بنام NoSE برای پیشنهاد طراحی شِمای بهینه پایگاه داده ستون‌گسترده کاساندرا معرفی شده است. این روش مبتنی بر هزینه، از فرموله‌سازی مساله با برنامه‌ریزی عدد صحیح صفر و یک[۱۹] برای نگاشت از مدل داده مفهومی پایگاه داده به شِمای پایگاه داده استفاده می‌کند. راهنمای NoSE شِمای مفهومی داده‌های برنامه و بارکاری را به عنوان ورودی دریافت می‌کند. سپس با استفاده از فرموله‌سازی مساله با برنامه‌ریزی عدد صحیح صفر و یک و حل آن، مجموعه بهینه‌ای از گروه‌های ستون که هزینه اجرای پرس‌وجوها را کمینه می‌کنند و یک مجموعه از طرح‌های اجرایی (یک طرح برای هر پرس‌وجو) را به عنوان خروجی پیشنهاد می‌دهد. با استفاده از شِمای پیشنهادی توسط ابزار NoSE به هر پرس‌وجوی توسط یک یا چند درخواست به پایگاه داده پاسخ داده می‌شود، به طوری که کل هزینه پاسخ‌دهی به پرس‌وجوها کمینه می‌گردد و اندازه کل گروه‌های ستون پیشنهادی محدود به یک فضای ذخیره‌سازی مشخص است. همچنین در این روش به منظور پشتیبانی از پرس‌وجوهای نوشتن، ابزار NoSE تعدادی پرس‌وجوی پشتیبان را برای بدست آوردن اطلاعات مورد نیاز برای اجرای عملیات نوشتن به شِمای طراحی شده اضافه می‌نماید.

نویسندگان در [۱۹] قسمتی از شِمای رابطه‌ای پایگاه داده تویتر را استفاده کرده‌اند و با استفاده از غیرنرمال‌سازی آن به طور خودکار به شناسایی شِمای ستون‌گرا NoSQL با کمترین هزینه پرداخته‌اند. این روش همانند روش [۱۸] با یک شِمای مفهومی شروع و بهینه‌سازی شِمای NoSQL را ارائه می‌دهد. همچنین واجک و همکارانش در تحقیقی مشابه با کار قبلی، الگوریتمی برای یافتن خودکار شِمای بهینه در پایگاه داده ستون‌گرا NoSQL ارائه نموده‌اند [۲۰]. این الگوریتم با دریافت مجموعه‌ای از پرس‌وجوهای تعریف شده و یک شِمای اولیه رابطه‌ای و با استفاده از روش غیرنرمال‌سازی شِما، به یافتن شِمای بهینه می‌پردازد. در این روش پرس‌وجوها با زبان قیود شی بیان می‌شوند که به راحتی قابل تبدیل به مدل مفهومی از رابطه‌ها هستند. در این الگوریتم ابتدا شِمای رابطه‌ای اولیه به طور خودکار غیرنرمال‌سازی می‌شود و تمام شِماهای ممکن (رابطه‌های غیرنرمال) ایجاد می‌شوند. سپس بر مبنای پرس‌وجوهای تعریف شده، اندیس‌های تک-ستونه برای جداول انتخاب می‌شوند. سرانجام با استفاده از یک تابع هزینه، هزینه اجرای پرس‌وجوها برای هر شِما محاسبه شده و شِمایی با کمترین هزینه به عنوان شِمای بهینه انتخاب می‌شود.

روش ارائه شده در [۲۱] به طراحی شِمای منطقی برای پایگاه داده تحلیلی در پایگاه داده ستون‌گرا NoSQL پرداخته است. در حقیقت این روش، پایگاه داده تحلیلی رابطه‌ای را به پایگاه داده تحلیلی ستون‌گرا NoSQL تبدیل می‌کند. این روش از تکنیک خوشه‌بندی K-means برای طراحی بهتر شِمای منطقی پایگاه داده ستون‌گرا(مجموعه‌ای از گروه‌های ستون) از جداول پایگاه داده تحلیلی، استفاده کرده است.

یانگ و همکارانش در [۲۲] برای یافتن شمای بهینه پایگاه داده ستون‌گرای Hbase برای پرس‌وجوهای OLAP، یک الگوریتم تکاملی جدید طراحی نموده‌اند. این روش بر اساس پرس‌وجوهای برنامه و با استفاده از الگوریتم تکاملی طراحی شده، ستون‌ها را گروه‌بندی می‌کند و شِمای بهینه‌ای از گروه‌های ستون را پیدا می‌کند. با استفاده از این روش کارایی خواندن پرس‌وجوهای OLAP (متوسط زمان پاسخ) به طور چشمگیری بهبود یافته است.

در [۲۳] مدلی برای طراحی شِما در پایگاه داده سندگرا NoSQL ارائه شده است. مدل پیشنهادی، شِمایی بر مبنای نیازمندی‌های سیستم و پارامترهایی نظیر موجودیت ها در مدل مفهومی و عملیات CRUD طراحی می‌کند. تمرکز این مدل بر ایجاد تعادل بین فاکتورهای مهم مدلسازی داده (نیازمندیهای سیستم) شامل ثبات، در دسترس پذیری و مقیاس پذیری است. شِمای ایجاد شده توسط این مدل امنیت و کارایی پرس‌وجوهای نوشتن-خواندن را بهبود می‌دهد.

همچنین کارهایی نظیر [۲۴،۲۵] به بهینه‌سازی شِماهای پایگاه داده NoSQL برای افزایش کارایی پرس‌وجوهای OLAP پرداخته‌اند. این روش‌ها یک مدل انبارداده را به مدل‌های NoSQL با هدف بهبود زمان پاسخ به پرس‌وجوها تبدیل کرده-اند.

تمامی کارهای مذکور، یک نگاشت از پیش تعریف شده ای را بر مبنای پرس‌جوهای برنامه ارائه می‌دهند. در این کارها نوشتن و اجرای پرس‌وجوها NoSQL وابسته به شِمای طراحی شده است. بنابراین با تغییر شِمای طراحی شده، نیاز به تلاش توسعه دهنده سیستم برای تغییر کدهای دسترسی به داده‌ها (پرس‌وجوها) است. در مقابل، تحقیقاتی نظیر [۲۹، ۳۰] یک روش مستقل از شِما را برای نوشتن و اجرای پرس‌وجوهای NoSQL ارائه می-دهند. این روش‌ها از اطلاعات نگاشت بین مدل مفهومی(نمودار ER) و شِمای NoSQL استفاده می‌کنند و به طور خودکار پرس-وجوهای بیان شده بر مبنای مدل مفهومی را به کدهای اجرایی برای یک زبان NoSQL خاص تبدیل می‌کنند. بنابراین شِماهایی

با انعطاف پذیری بالاتری را تولید می‌کنند و اجازه می‌دهند که بدون نیاز به تغییر کدهای دسترسی به داده‌ها، شِمای پایگاه داده تغییر کند.

شایان ذکر است که در کارهایی نظیر [۲۸-۲۶] به طراحی شِمای پایگاه داده NoSQL بدون در نظر گرفتن بارکاری پرداخته اند. این روش‌های با نادیده گرفتن تحلیل بارکاری، فقط قوانین نگاشتی برای تبدیل مدل مفهومی به مدل فیزیکی پایگاه داده NoSQL ارائه کرده‌اند. بنابراین در این روش‌ها بهینه سازی شِما پایگاه داده نادیده گرفته شده است.

همانطور که مشاهده شد، هدف اصلی اکثر تحقیقات انجام شده در زمینه خودتنظیمی شِمای پایگاه داده NoSQL، بهینه سازی شِما برای بارکاری برنامه و انتخاب یک شِمای مناسب است. تمایز اصلی بین این روش‌ها در قوانین نگاشت و راه‌حل ارائه شده برای بهینه‌سازی شِما است. اکثر راهکارهای ارائه شده در طراحی شِمای بهینه‌شده برای پایگاه داده ستون‌گسترده NoSQL فقط به بهینه‌سازی پرس‌وجوهای خواندن پرداخته‌اند. در این روش‌ها یا به چگونگی پاسخ به پرس‌وجوهای نوشتن اشاره‌ای نشده است و یا پرس‌وجوهای نوشتن برمبنای شِمای بدست آمده از بهینه-سازی پرس‌وجوهای خواندن مدیریت شده‌اند. فقط ابزار NoSE به بهینه‌سازی شِمای پایگاه داده ستون‌گسترده برای کل بارکاری برنامه توسط ارائه یک روش مبتنی بر هزینه و فرموله‌سازی مسئله بهینه سازی با برنامه‌ریزی عدد صحیح باینری (BIP) پرداخته است. بنابراین در اکثر کارهای ارائه شده شِما پایگاه داده برای پرس‌وجوهای نوشتن بهینه نشده است و بنابراین عملیات نوشتن پرتکرار سربار بالایی را به پایگاه داده تحمیل می‌کنند. روش پیشنهادی ما، همانند ابزار NoSE [۱۸]، به بهینه‌سازی شِمای پایگاه داده برای کل بارکاری برنامه (هم پرس‌وجوهای خواندن و هم پرس‌وجوهای نوشتن) می‌پردازد. اگرچه آن یک راه‌حل متفاوتی را ارائه می‌دهد، به طوریکه سربار ذخیره سازی و سربار اجرای پرس‌وجوهای نوشتن را کاهش می‌دهد.

جدول ۱ مقایسه‌ای بین روش ارائه شده در این مقاله با برخی از مهمترین تحقیق‌های انجام شده در این زمینه را نشان می‌دهد.

جدول ۱: مقایسه بین روش پیشنهای و تحقیق‌های انجام شده.

تحقیق	مدل پایگاه داده NoSQL	روش طراحی شما	نوع بارکاری	نوع تراکنش
[۱۶]	ستون‌گسترده	مبتنی بر تحلیل بارکاری	OLTP	فقط خواندنی
[۱۷]	سندگرا	مبتنی بر تحلیل بارکاری	OLTP	فقط خواندنی
[۱۸]	ستون‌گسترده	مبتنی بر تحلیل بارکاری	OLTP	خواندن و نوشتن
[۱۹]	ستون‌گسترده	مبتنی بر تحلیل بارکاری	OLTP	فقط خواندنی
[۲۴]	ستون‌گسترده	مبتنی بر تحلیل بارکاری	OLAP	فقط خواندنی
[۲۵]	سندگرا	مبتنی بر تحلیل بارکاری	OLAP	فقط خواندنی
[۲۶]	سندگرا	نادیده گرفتن تحلیل بارکاری	OLTP	فقط خواندنی
[۲۸]	گرافی	نادیده گرفتن تحلیل بارکاری	OLTP	فقط خواندنی
روش مقاله	ستون‌گسترده	مبتنی بر تحلیل بارکاری	OLTP	خواندن و نوشتن

۴- روش پیشنهادی

در این بخش، روشی مبتنی بر بارکاری برای انتخاب خودکار شِمای بهینه‌یافته از پایگاه داده ستون‌گسترده NoSQL ارائه شده است. این روش شامل فرایند نگاشت مدل داده مفهومی به مدل داده منطقی و پیاده سازی فیزیکی آن در کاساندرا است. در این روش از اطلاعات بارکاری برای رسیدن به کارایی بهتر پرس وجوها استفاده شده است که با کمینه کردن تعداد درخواست‌ها به پایگاه داده منجر به طراحی شِمای بهینه‌یافته می‌شود.

منطق بهینه‌سازی شِمای پایگاه داده NoSQL غیرنرمال‌سازی و افزونی داده برای بالا بردن کارایی پرس‌وجوهای خواندن است. بطوریکه هر پرس‌وجوی خواندن فقط توسط یک درخواست به پایگاه داده پاسخ داده شود. برای رسیدن به این هدف باید برای هر پرس‌وجوی خواندن یک جدول جهت بازیابی مستقیم رکوردهای پرس‌وجو در پایگاه داده ستون‌گسترده ایجاد شود. ایجاد یک جدول برای هر پرس‌وجوی خواندن سرعت و کارایی اجرای پرس‌وجوهای خواندن را بالا می‌برد ولی منجر به تعداد جدول‌های زیاد و کپی‌های تکرار ویژگی‌های موجودیت‌ها در جداول‌های مختلف می‌شود که نتیجه آن افزایش سربار ذخیره-سازی است. همچنین با در نظر گرفتن پرس‌وجوهای نوشتن در روی این جدول‌ها، هزینه نگهداری شِمایی با یک جدول برای هر پرس‌وجو خواندن ممکن است بسیار بالا رود. زیرا اجرای عملیات نوشتن شامل تعداد درخواست‌های بیشتر به پایگاه داده و پرهزینه است. بنابراین پرس‌وجوهای نوشتن بر خلاف پرس-وجوهای خواندن، سطح غیرنرمالسازی را محدود می‌کنند.

ایده اصلی در روش ارائه شده، بهینه‌سازی شِمای پایگاه داده برای کل بارکاری برنامه یعنی پرس‌وجوهای خواندن و نوشتن است. یک راه‌حل برای این مساله، کنترل توازن بین نرمالسازی و

غیرنرمالسازی در طراحی شِما است. برای رسیدن به این هدف، روش پیشنهادی با در نظر گرفتن اطلاعات مربوط به وزن هر پرس‌وجو در بارکاری و تعیین الویت طراحی شِما برای پرس-وجوهای بارکاری می‌پردازد تا از غیرنرمال سازی برای داده‌هایی که مکررا به‌روز رسانی و به ندرت خوانده می‌شوند، جلوگیری کرده و همچنین به‌روزرسانی‌های پرهزینه برای پرس‌وجوهای نوشتن کم تکرار را انجام دهد. بنابراین شِمای خودکار تولید شده توسط روش پیشنهادی با کمینه کردن تعداد درخواست‌های ارسالی برای کل بارکاری برنامه به پایگاه داده منجر به کارایی مناسبی از بارکاری می‌شود.

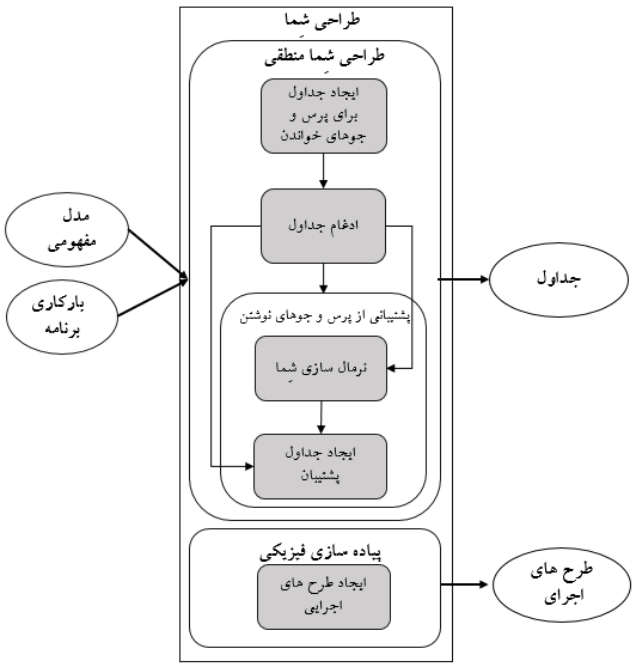
در ادامه این بخش، ابتدا دید مفهومی از روش پیشنهادی ارائه می‌شود. سپس مراحل روش پیشنهادی شامل «طراحی شِما منطقی» و «پیاده سازی فیزیکی» در جزئیات بیان می‌شوند.

۴-۱ دید کلی از روش پیشنهادی

شکل ۲ دید کلی از روش پیشنهادی را نشان می‌دهد. در این روش فرایند طراحی شِما به دو مرحله «طراحی شِما منطقی» و «پیاده سازی فیزیکی» تقسیم شده است. مطابق این شکل مدل داده مفهومی و اطلاعات بارکاری که توسط طراح برنامه مشخص شده است به عنوان ورودی مرحله «طراحی شِما منطقی» هستند. در این مرحله با استفاده از تحلیل اطلاعات بارکاری، مدل داده مفهومی به مدل داده منطقی ستون‌گسترده تبدیل و مجموعه‌ای از جداول ایجاد می‌شود. در حقیقت، هدف اصلی این مرحله پیشنهاد یک مجموعه بهینه‌یافته‌ای از جداول برای پاسخ-دهی به پرس‌وجوها و اطلاعاتی است که باید در هر جدول ذخیره شود. سپس در مرحله اجرا جداول پیشنهاد شده از مرحله طرح‌ریزی در کاساندرا ایجاد می‌شوند و مجموعه‌ای از طرح‌های اجرایی پرس‌وجوها (یک طرح برای هر پرس‌وجو) تولید می‌شود. هر طرح شرح می‌دهد که چگونه برنامه باید از جداول در شِمای پیشنهادی برای اجرای یک پرس‌وجو استفاده کند.

در ادامه به توصیف مفاهیم مرتبط با مساله طراحی شِما در پایگاه داده ستون‌گسترده می‌پردازیم:

مدل مفهومی: مدل داده مفهومی یک توصیف سطح بالا از اطلاعاتی است که در پایگاه داده ذخیره خواهند شد. شمای مفهومی در نظر گرفته شده در این تحقیق، نوع محدود شده‌ای از نمودار موجودیت-ارتباط [۳۱] است که شامل موجودیت‌ها و ارتباط انجمنی بین آنها و نوع اتصال در این ارتباط(یک-یک، یک-چند، چند-چند) است.



شکل ۲: دید کلی از روش پیشنهادی

بارکاری: بارکاری برنامه شامل مجموعه‌ای از جملات پرس-وجوهای خواندن و نوشتن است. هر پرس‌وجوی خواندن مقادیر ویژگی‌ها از یک یا چند موجودیت در مدل مفهومی را برمی-گرداند. همچنین پرس‌وجوهای نوشتن عملیات درج، حذف و بروزرسانی را بر روی موجودیت‌ها انجام می‌دهند. به هر دستور پرس‌وجو در بارکاری، یک وزن مشخصی نسبت داده می‌شود که برابر با فراوانی نسبی[۲۰] (تعداد تکرار) آن دستور در بارکاری است. در این تحقیق پرس‌وجوهای بارکاری تحت مدل مفهومی و ساختار CQL-like بیان می‌شوند. زیرا دستورات زبان CQL در روی جداول ایجاد شده، قابل اعمال هستند که در ابتدا مشخص نمی‌باشند و هدف اصلی حلقه کنترل ارائه شده، پیشنهاد مجموعه بهینه‌یافته‌ای از جداول برای پرس‌وجوهای بارکاری برنامه است.

طراحی شِما در پایگاه داده ستون‌گسترده: طراحی شِما در پایگاه داده ستون‌گسترده (ستون‌گرا) که نوع خاصی از پایگاه داده NoSQL است، تعریف جداولی است که هر جدول توسط کلیدهای پارتیشن، کلیدهای خوشه‌بندی و ستون‌های غیرکلیدی بکار رفته در آن ارائه می‌شود. طراحی شِما در پایگاه داده ستون گسترده، شامل تعیین مجموعه‌ای از جداول و اطلاعاتی است که در هر جدول برای یک بارکاری مشخص باید ذخیره شود تا هزینه اجرای پرس‌وجوهای بارکاری کمینه گردد. بنابراین مساله طراحی شِمای پایگاه داده ستون گسترده به صورت زیر تعریف می‌شود:

«با داشتن یک مدل مفهومی و بارکاری برنامه، مجموعه بهینه-یافته‌ای از جداول انتخاب شود که با کمینه کردن تعداد درخواست‌ها به پایگاه داده منجر به کارایی مناسبی از بارکاری گردد، در حالیکه سربار ذخیره سازی تا حد ممکن کاهش یابد.»

۴-۲ طراحی شِمای منطقی

طراحی شِمای منطقی شامل فرایند نگاشت مدل داده مفهومی به مدل داده منطقی برمبنای بارکاری برنامه است که منجر به تولید مجموعه‌ای از جداول بهینه‌شده برای بالا بردن کارایی پرس-وجوهای بارکاری می‌شود. این فرایند شامل سه مرحله اصلی است. مرحله اول شامل فرایند بهینه‌سازی شِما برای پرس-وجوهای خواندنی است. در این مرحله با استفاده از غیرنرمالسازی، برای هر پرس‌وجو خواندنی در بارکاری یک جدول برای بازیابی مستقیم پاسخ آن تعریف می‌شود. همانطور که گفته شد ایجاد یک جدول برای هر پرس‌وجوی خواندن منجر به تعداد جداول زیاد و تکرار ویژگی‌های موجودیت‌ها در جداول مختلف می‌شود که نتیجه آن افزایش سربار ذخیره‌سازی و سربار اجرای پرس‌وجوهای نوشتن است. همچنین بعضی از جداول ممکن است زیرمجموعه‌ای از جداول دیگر یا ساختاری مشابه با آنها داشته باشند. بنابراین در مرحله دوم قوانینی برای ادغام جداول ایجاد شده تا حد امکان ارائه می‌شود. مرحله سوم به تغییرات شِمای پیشنهادی برای پشتیبانی از پرس‌وجوهای نوشتن می-پردازد. الگوریتم ۱ فرایند نگاشت سه مرحله‌ای برای ایجاد جداول از بارکاری برنامه را نشان می‌دهد.

Algorithm 1. create tables for workload

Input: A Workload where contains a set of read-queries $\boldsymbol{RQ}$ and write-queries $\boldsymbol{WQ}$, A Conceptual schema $\boldsymbol{ER}$ where contains a set of entities $\boldsymbol{E}$ and relationships $\boldsymbol{R}$.
Output: A set of tables $\boldsymbol{T}$.

// **Step 1** *create tables for read-queries*
1. **for** each read − query rq *in* RQ **do**
2. $T = T \cup$ Denormalize(rq, ER)
3. **end for**
// **Step 2** *merge tables*
4. $T = merge(T)$
// **Step 3** *support write-queries*
5. $T' = T$
6. **for** each write − query wq *in* WQ **do**
7. **for** each table t *in* T' **do**
8. **if** effect $(wq.t)$ **then**
9. **if** $|Entities(t)| \neq 1$ *and* $Entity(wq).attrs \subseteq t.Cols$ and $wq.weight > all\ of\ rq(t).weight$ **then**
// **Step 3.1** *normalize table*
10. $T = T \cup normalize(wq, t)$
11. **else**
// **Step 3.2** *create support table*
12. $T = T \cup SupportWQ(wq, t)$
13. **end if**
14. **end if**
15. **end for**
16. $T = merge(T)$
17. return T

در ادامه جزئیات هر مرحله و الگوریتم مربوط به آن را با مثال سیستم RUBIS شرح داده می‌شود.

مرحله ۱) ایجاد جداول برای پرس‌وجوهای خواندن: در این مرحله به تحلیل دستورات SELECT پرداخته می‌شود و برای هر پرس‌وجوی خواندن یک جدول تعریف می‌شود که با استفاده از آن جدول می‌توان به پرس‌وجو فقط با ارسال یک درخواست به پایگاه داده پاسخ داد. الگوریتم ۲ چگونگی ایجاد یک جدول برای هر پرس‌وجو را شرح می‌دهد. لازم به ذکر است که در پایگاه داده ستون‌گسترده کاساندرا بازیابی داده‌ها کاملا وابسته به کلید اصلی است و روی فیلدهای غیرکلید نمی‌توان جستجو انجام داد، همچنین هر درخواست خواندن معتبر به پایگاه داده حداقل باید شامل یک جمله شرطی مساوی باشد. از این‌رو در این الگوریتم ابتدا برای هر جمله شرطی در عبارت WHERE اگر عملگر شرط "مساوی" باشد، ویژگی مربوط به آن به کلید پارتیشن و در غیر اینصورت به کلید خوشه‌بندی نگاشت می‌شود (خطوط ۱-۷). همچنین ویژگی‌های مرتب سازی که در عبارت ORDER BY بکاررفته‌اند به کلید خوشه‌بندی نگاشت می‌شوند (خطوط ۸-۱۰). سپس برای شناسایی یکتای ردیف‌ها در جدول ایجاد شده باید کلید اصلی موجودیت‌های بکار رفته در عبارت FROM به کلید خوشه‌بندی اضافه شوند (خطوط ۱۱-۱۵). در پایان ویژگی-های انتخابی در دستور SELECT به ستون‌های غیرکلیدی در جدول نگاشت می‌شوند (خطوط ۱۶-۲۰).

Algorithm 2. Create tables for read-queries (Denormalize)

Input: A Read-query $\boldsymbol{Q}$, A Conceptual schema $\boldsymbol{ER}$ where contains a set of entities $\boldsymbol{E}$ and relationships $\boldsymbol{R}$.
Output: A table $\boldsymbol{t}$ $[Prtition\_Key][Cluster\_Key][\ nonkey\_Columns]$ for $\boldsymbol{Q}$, $Entities\ (\boldsymbol{t})$. $Relationships\ (\boldsymbol{t})$.

1. **for** each search_condition statement sc *in* $Q.WHERE\ clause$ **do**
// **Equality Filter Attributes**
2. **if** $sc.operation = '='$ **then**
3. $Prtition\_Key = Prtition\_Key \cup sc.attr$
// **Inequality Filter Attributes**
4. **else**
5. $Cluster\_Key = Cluster\_Key \cup sc.attr$
6. **end if**
7. **end for**
// **Ordering Attributes**
8. **for** each attribute $attr$ *in* $Q.ORDER\_BY\ clause$ **do**
9. $Cluster\_Key = Cluster\_Key \cup attr$
10. **end for**
// **Primary Key Attributes**
11. **for** each entity $e \in \boldsymbol{E}$ *in* $Q.FROM\ clause$ **do**
12. **if** $e.primary\_key \notin \{Prtition\_Key \cup Cluster\_Key\}$ **then**

13. $Cluster\_Key = Cluster\_Key \cup e.primary\_key$
14. **end if**
15 **end for**
// Select Attributes
16. **for** each $attr\ in\ Q.SELECT\ clause$ **do**
17. **if** $attr \notin Prtition\_Key \cup Cluster\_Key$ **then**
18. $nonkey\_Columns = nonkey\_Columns \cup attr$
19. **end if**
20. **end for**
// Identify entities and relationships for table
21. **for** each entity $e \in \boldsymbol{E}\ and\ r \in \boldsymbol{R}\ in\ Q.FROM\ clause$ **do**
22. $Entities(t) = Entities(t) \cup e$
23. $Relationships(t) = Relationships(t) \cup r$
24. **end for**
25.return $t[Partition\_Key][Cluster\_Key][nonkey\_Columns]$, $Entities(t), Relationships(t)$

مرحله ۲) ادغام جداول: شِمای ایجاد شده از مرحله قبل شامل جداول بسیاری است که بعضی از جداول ممکن است زیرمجموعه‌ای از جداول دیگر یا ساختاری مشابه با آنها داشته باشند. بنابراین در این مرحله قوانینی برای ادغام جداول ایجاد شده تا حد امکان ارائه می‌شود. جداول حاصل شده از این مرحله قادر به پاسخگویی به بیش از یک پرس‌وجو هستند، در حالیکه سربار ذخیره سازی و سربار اجرای پرس‌وجوهای نوشتن را کاهش می‌دهند. در ادامه هر یک از این قوانین شرح داده می‌شود.

- قانون اول: اگر جدول t_1 زیر مجموعه جدول t_2 باشد، آنگاه میتوان جدول t_1 را حذف کرد و به پرس‌وجوی مربوط به آن با استفاده از جدول t_2 پاسخ داد. جدول t_1 زیر مجموعه جدول t_2 است اگر (۱) تمام ستون‌های جدول t_1 زیر مجموعه ستون‌های جدول t_2 باشند، و (۲) تمام ارتباط‌های بین موجودیت‌ها از جدول t_1 زیر مجموعه ارتباط‌ها از جدول t_2 است، و (۳) کلید خوشه‌بندی از جدول t_1 زیر مجموعه کلید خوشه‌بندی جدول t_2 باشد، و (۴) کلید پارتیشن جدول t_1 و جدول t_2 یکسان باشد.

- قانون دوم: اگر ساختار جدول t_1 و جدول t_2 مشابه یکدیگر باشد، آنگاه می‌توان یکی از این دو جدول را حذف کرد و به پرس‌وجوی مربوط به جدول حذفی با استفاده از ایجاد دید ذخیره شده در روی جدول دیگر پاسخ داد. این بدین معناست که اگر ویژگی‌های موجودیت‌ها (ستون‌ها) و ارتباط‌های بین موجودیت‌ها از جدول t_1 زیر مجموعه جدول t_2 باشند و فقط کلید اصلی آنها در روی ستون‌های متفاوتی تعریف شده باشد، آنگاه می‌توان جدول t_1 را حذف کرد و یک دید ذخیره شده در روی جدول t_2 ایجاد کرد. در دید ایجاد شده، ستون‌های کلید پارتیشن و خوشه‌بندی جدول t_1 که در جدول t_2 نیستند به آن اضافه می‌شوند.

مرحله ۳) پشتیبانی از پرس‌وجوهای نوشتن: هر پرس‌وجوی نوشتن نظیر UPDATE، DELETE یا INSERT فقط یک موجودیت در شِمای مفهومی را تغییر می‌دهد. بنابراین انتظار می‌رود که فقط با یک درخواست به پایگاه داده اجرا شوند. از آنجایی که بهینه‌سازی برای پرس‌وجوهای خواندن منجر به غیرنرمال‌سازی شِما می‌شود، چنین چیزی به ندرت اتفاق می‌افتد. مگر اینکه برای بهینه‌سازی پرس‌وجوی نوشتن، شِما را نرمال شده طراحی کنیم. همانطور که گفته شد هدف اصلی این تحقیق بهینه‌سازی شِما برای کل بارکاری برنامه است که شامل مجموعه‌ای از پرس‌وجوهای خواندن و نوشتن است. بنابراین در این روش با تخصیص وزن به هر پرس‌وجو که نشان دهنده میزان تکرار آن در بارکاری است، اولویت طراحی شِما برای پرس‌وجوهای بارکاری مشخص می‌گردد تا بتوان شِمای بهینه‌شده‌ای برای بارکاری ورودی طراحی کرد. از این‌رو اگر در بارکاری ورودی، وزن درخواست نوشتن در یک موجودیت کوچکتر یا مساوی با وزن درخواست خواندن از آن باشد، الویت با خواندن است و از روش ایجاد جداول پشتیبان برای پاسخ به دستورات نوشتن استفاده می‌کنیم. اما اگر برعکس این باشد، الویت با نوشتن است و باید با استفاده از روش نرمالسازی، ویژگی‌های مربوط به آن موجودیت را در یک جدول مجزا قرار دهیم تا سرعت اجرای دستور نوشتن بالا رود و تنها با یک درخواست به پایگاه داده عمل نوشتن انجام شود.

مرحله ۳.۱) ایجاد جداول پشتیبان برای پرس‌وجو نوشتن: در مساله طراحی شِما، پرس‌وجوهای نوشتن دو دسته متمایز بررسی می‌شوند. دسته اول دستورات UPDATE و DELETE است که در روی رکوردهای موجود اثر می‌گذارند. دسته دوم دستورات INSERT است که یک رکورد جدید را درج می‌کنند.
دستورات UPDATE و DELETE: برای تحلیل و پاسخ‌گویی به هر یک از این پرس‌وجوها، ابتدا جداولی که توسط این پرس‌وجوها تحت تاثیر هستند مشخص می‌شوند. برای اجرای این دستورات، باید تمام ستون‌های کلید اصلی در شرط WHERE مشخص شده باشند، در غیر این صورت باید مقادیر آنها را بدست آوریم. سپس برای هر یک از این جداول بررسی می‌شود که آیا کلید اصلی آن شامل ویژگی‌های جستجو در عبارت WHERE از پرس‌وجوی مربوطه است یا خیر. اگر این امکان برای جدول باشد، آنگاه دستور UPDATE یا DELETE بر روی آن اجرا می‌

شود و نیازی به ایجاد جدول پشتیبان نیست. در غیر اینصورت یک جدول پشتیبان برای شناسایی رکوردهایی که باید UPDATE یا DELETE شوند ایجاد می‌شود. سپس با انجام یک درخواست خواندن در روی جدول پشتیبان می‌توان اطلاعاتی که برای اجرای دستور UPDATE یا DELETE نیاز است را بدست آورد. خطوط ۱-۷ از الگوریتم ۳ جزئیات روند این کار را نشان می‌دهد. همچنین چگونگی ایجاد جداول پشتیبان برای دستور UPDATE یا DELETE در الگوریتم ۴ نشان داده شده است.

Algorithm 3. Support write-query

Input: A write-query $\boldsymbol{wq}$, A table $\boldsymbol{t}$.
Output: A set of support tables $\boldsymbol{ST}$
1. **if** $wq.type \in \{Update, Delete\}$ **then**
2 **for each** search_condition statement sc $in\ wq.WHERE\ clause$ **do**
3. **if** $sc.attr \notin t.Primary\_Key$ **then**
4. $ST = SuppTable\_update\&delete(wq, t)$
5. return ST
6. **end if**
7. **end for**
8. **elseif** $wq.type = Insert$ **then**
9. **if** $|Entities(t)| \neq 1$ **then**
10. $ST = SuppTable\_insert(wq, t)$
11. return ST
12. **end if**
13. **end if**
14. return ϕ

Algorithm 4. Create support table for update/delete query

Input: A update/delete query $\boldsymbol{wq}$, A table $\boldsymbol{t}$.
Output: A support table for $\boldsymbol{wq}$.
1. **for** each search_condition statement sc $in\ wq.WHERE\ clause$ **do**
// **Equality Filter Attributes**
2. **if** $sc.operation = '='$ **then**
3. $Prtition\_Key = Prtition\_Key \cup sc.attr$
// **Inequality Filter Attributes**
4. **else**
5. $Cluster\_Key = Cluster\_Key \cup sc.attr$
6. **end if**
7. **end for**
8. **for each** $attr\ in\ t.primary\_key$ **do**
9. **if** $attr \in \{e.primary\_key \mid e \in Entities(t)\}$ **then**
10. $Cluster\_Key = Cluster\_Key \cup attr$
11. **else**
12. $nonkey\_Columns = nonkey\_Columns \cup attr$
13. **end if**
14. **end for**
15. **return** $[Partition\_Key][Cluster\_Key][nonkey\_Columns]$

دستورات INSERT: یک دستور INSERT رکورد جدیدی را در یک موجودیت درج می‌کند و منجر به درج در جداولی می‌شود که شامل ویژگی‌های آن موجودیت هستند. اگر جدول فقط شامل ویژگی‌های موجودیت مربوطه باشد، رکورد جدید به سادگی در آن درج می‌شود. ولی اگر جدول شامل ویژگی‌هایی از چند موجودیت متفاوت باشد، درج رکورد جدید در آن نیاز به ایجاد یک یا بیشتر جدول پشتیبان برای یافتن مقادیر ویژگی-های موجودیت‌های دیگر در آن است. خطوط ۱۳-۸ از الگوریتم ۳ جزئیات روند این کار را نشان می‌دهد. همچنین چگونگی ایجاد جداول پشتیبان برای دستور INSERT در الگوریتم ۵ نشان داده شده است.

Algorithm 5. Create support table for insert-query

Input: A insert-query $\boldsymbol{wq}$, A table $\boldsymbol{t}$.
Output: A set of support table $\boldsymbol{ST}$ for $\boldsymbol{wq}$.
1. $SCF = \emptyset$
2. **if** $Relationships(wq.LINKTO) \cap Relationships(t) = \emptyset$ **then** return ϕ
3. $E = Entities(wq.LINKTO)$
4. $e' = Entity(wq.INTO)$
5. $PK = \{e.primary\_key \mid e \in E\}$
6. A= $\{t.cols\} - \{PK \cup e'.attrs\}$
7. **if** A= ϕ **then** return ϕ
8. **for each** $e\ in\ E$ **do**
9. **for each** $attr\ in$ A **do**
10. **if** $attr.Entity = e\ \ or\ \ relationship(attr.Entity.e) \subseteq Relationships(t)$ **then**
11. $Prtition\_Key = e.primary\_key$
12. **if** $attr \in \{e^{pk}.primary\_key \mid e^{pk} \in Entities(t)\}$ **then**
13. $Cluster\_Key = Cluster\_Key \cup attr$
14. **else**
15. $nonkey\_Columns = nonkey\_Columns \cup attr$
16. **end if**
17. delete_attr (A, $attr$)
18. **end if**
19. **end for**
20. $ST = ST \cup \{[Partition\_Key][Cluster\_Key][nonkey\_Columns]\}$
21. **end for**
22. return ST

مرحله ۳.۲) نرمالسازی شِما: هدف از نرمالسازی کاهش سربار ذخیره‌سازی و بهینه‌سازی شِما برای کارایی نوشتن است. این روش ویژگی‌های مربوط به موجودیتی که عملیات نوشتن در روی آن انجام می‌شود را در یک جدول مجزا قرار می‌دهد تا سرعت اجرای دستور نوشتن بالا رود و با درخواست‌های کمتری به پایگاه داده عمل نوشتن انجام شود.

الگوریتم ۶ جزئیات روند این کار را نشان می‌دهد. ورودی این الگوریتم، پرس‌وجوی نوشتن و جدولی است که تحت تاثیر این پرس‌وجو است. در این الگوریتم ابتدا ویژگی‌های مربوط به موجودیتی که عملیات نوشتن در روی آن انجام می‌شود از جدول مربوطه (t) حذف می‌شوند و در یک جدول مجزا (t^n) قرار می-گیرند. در صورتیکه عمل نوشتن update یا delete باشد، به-روزرسانی یا حذف به طور مستقیم فقط بر روی جدول ایجاد

شده (t^n) انجام می‌شود. ولی دستور نوشتن insert که دارای عبارت LINKTO است علاوه بر درج رکورد جدید از موجودیت درجی در جدول t^n باید کلید اصلی موجودیت درجی در جدول t برای اتصال به کلید اصلی موجودیت LINKTO، درج شود.

Algorithm 6. normalize

Input: A write-query wq, A table t.
Output: A normalized table t^n.

1. **for** each $attr\ in\ t.Cols$ **do**
2. **if** $attr \in Entity(wq).attrs$ **then**
3. **if** $attr \in Entity(wq).primary\_key$ **then**
4. $t^n.Prtition\_Key = attr$
5. **elseif** $attr \in t.nonkey\_Columns$ **then**
6. delete_attr($t.nonkey\_Columns.attr$)
7. $t^n.nonkey\_Columns = t^n.nonkey\_Columns \cup attr$
8. **elseif** $attr \in t.Cluster\_Key$ then
9. delete_attr($t.Cluster\_Key.attr$)
10. $t^n.Cluster\_Key = t^n.Cluster\_Key \cup attr$
11. **end if**
12. **end if**
13. **end for**
14. return t^n

۳-۴ پیاده‌سازی فیزیکی

در مرحله پیاده‌سازی فیزیکی، جداول پیشنهاد شده از مرحله «طراحی شِما منطقی» در محیط کاساندرا ایجاد می‌شوند و سپس مجموعه‌ای از طرح‌های اجرایی پرس‌وجوها (یک طرح برای هر پرس‌وجو) تولید می‌شود. هر طرح شرح می‌دهد که چگونه برنامه باید از جداول در شِمای پیشنهادی برای اجرای یک پرس‌وجو استفاده کند. در واقع، در این مرحله طراح برنامه از جداول پیشنهادی در مرحله قبل برای ایجاد طرح‌های اجرایی به‌صورت دستی با استفاده از دستورات CQL که در کاساندرا قابل اجرا هستند، استفاده می‌کند. تولید خودکار طرح‌های اجرایی به کارهای آینده سپرده شده است.

۵- ارزیابی

با توجه به تمرکزی که مطالعات حوزه خودکارسازی طراحی شِمای پایگاه داده NoSQL در این اواخر داشته‌اند، مورد مطالعه سیستم‌های تجارت الکترونیک تحت وب نظیر وب سایت حراجی RUBIS [۲۱] به نظر مناسب می‌رسد.

در این تحقیق سیستم پایگاه داده کاساندرا که یک پایگاه داده ستون‌گرا NoSQL است برای پیاده‌سازی و ارزیابی روش پیشنهادی در نظر گرفته شده است. برای تطبیق پایگاه داده رابطه‌ای RUBIS با کاساندرا، یک مدل مفهومی بر مبنای موجودیت‌های آن ایجاد و سپس پرس‌وجوهای بارکاری تحت مدل مفهومی و ساختار CQL-like بیان شده‌اند. مطالعه موردی RUBIS یک برنامه محک تحت وب است که عملکرد یک سایت حراجی مانند Ebay را شبیه‌سازی می‌کند. تعاملات اصلی کاربران در این سایت حراج شامل فروش، جستجو و پیشنهاد قیمت است. مدل مفهومی RUBIS شامل هفت موجودیت کاربران، کالاها، دسته‌ها، نواحی، مبلغ پیشنهادی، خرید فوری و نظرات و ارتباط‌های بین آنها است. تعاملات RUBIS شامل دو ترکیب بارکاری browsing و bidding است که تراکنش‌های آنها بر طبق پروژه اصلی RUBIS وزن‌دهی شده‌اند. ترکیب بارکاری browsing فقط شامل تراکنش‌های خواندن از پایگاه داده است، در حالیکه bidding شامل تراکنش‌های خواندن از و نوشتن در پایگاه داده است. در بارکاری bidding با تغییر وزن تراکنش-هایی که شامل نوشتن هستند می‌توان توزیع‌های بارکاری جدیدی را تعریف کرد. در این تحقیق برای آزمایش و ارزیابی راهکار پیشنهادی، دو ترکیب بارکاری دیگر نظیر light bidding و heavy bidding تعریف می‌شود. در بارکاری light bidding وزن تراکنش‌هایی که شامل نوشتن هستند توسط ضریب ۱/۱۰۰ کاهش یافته است و در بارکاری heavy bidding وزن تراکنش-هایی که شامل نوشتن هستند توسط ضریب ۱۰۰ افزایش یافته است. تراکنش‌ها و وزن آنها برای ترکیب‌های مختلف bidding در جدول ۲ نشان داده شده است.

برای ارزیابی ابتدا توسط مرحله «طراحی شِمای منطقی» شِما برنامه RUBIS به طور خودکار تولید می‌شود. سپس طراح برنامه با استفاده از جداول پیشنهادی در شِما، طرح‌های اجرایی پرس-وجوها را بر طبق دستورات CQL به صورت دستی ایجاد می‌کند. سرانجام شِمای پیشنهادی و طرح‌های اجرایی در محیط کاساندرا پیاده سازی و معیارهای ارزیابی اندازه گیری می‌شوند. برای انجام آزمایشات، شِمای پیشنهادی از روش ارائه شده با شِمای نرمال که به صورت دستی با سطح نرمالسازی بالا طراحی شده است و شِمای بدست آمده از ابزار NoSE [۱۸] مقایسه می‌شود. ارزیابی روش پیشنهادی شامل بررسی کیفیت و کارایی شِمای پیشنهادی است. معیار متوسط زمان اجرا[۲۲] برای بارکاری ورودی جهت ارزیابی کیفیت شِمای پیشنهادی و معیار زمان تولید شما[۲۳] جهت ارزیابی کارایی الگوریتم تولید شِما سنجیده می‌شود.

جدول۲: ترکیب‌های مختلف بارکاری bidding

Transactions	statements	Light Bidding		Bidding		Heavy Bidding	
		Weight	Count	Weight	Count	Weight	Count
BrowseCategories	SELECT	7.65	6617	7.65	6617	7.65	6617
ViewBidHistory	SELECT	1.54	1335	1.54	1335	1.54	1335
ViewItem	SELECT	14.17	12265	14.17	12265	14.17	12265
SearchItemsByCategory	SELECT	15.96	13790	15.96	13790	15.96	13790
ViewUserInfo	SELECT	2.48	2150	2.48	2150	2.48	2150
RegisterItem	INSERT	0.53*1/100	5	0.53	462	0.53*100	46200
RegisterUser	INSERT	1.07*1/100	9	1.07	929	1.07*100	92900
BuyNow	SELECT	1.16	1005	1.16	1005	1.16	1005
StoreBuyNow	SELECT, UPDATE, INSERT	1.10*1/100	10	1.10	956	1.10*100	95600
PutBid	SELECT	5.40	4670	5.40	4670	5.40	4670
StoreBid	SELECT,UPDATE, INSERT	3.74*1/100	32	3.74	3237	3.74*100	323700
PutComment	SELECT	0.46	402	0.46	402	0.46	402
StoreComment	SELECT,UPDATE, INSERT	0.45*1/100	4	0.45	389	0.45*100	38900
AboutMe	SELECT	1.71	1477	1.71	1477	1.71	1477
SearchItemsByRegion	SELECT	6.34	5489	6.34	5489	6.34	5489
BrowseRegions	SELECT	2.27	1963	2.27	1963	2.27	1963

ارزیابی روش پیشنهادی شامل بررسی کیفیت و کارایی شِمای پیشنهادی است. معیار متوسط زمان اجرا[۲۴] برای بارکاری ورودی جهت ارزیابی کیفیت شِمای پیشنهادی و معیار زمان تولید شما[۲۵] جهت ارزیابی کارایی الگوریتم تولید شِما سنجیده می‌شود. در این تحقیق، همچنین به منظور ارزیابی کیفیت شِما پیشنهادی دو معیار دیگر تعریف شده است:

معیار تکرار دسترسی (AF): این معیار نشان دهنده میزان دفعات دسترسی به هر جدول در شمای طراحی شده است و به صورت رابطه (۱) تعریف می‌شود.

$$AF(t) = \sum_{i=1}^{n} \mathrm{V}(q_i \,.\, t) * \ Count_{q_i} \qquad (۱)$$

n تعداد پرس‌وجوهایی است که از جدول t استفاده کرده‌اند. تابع V برابر حجم نمونه هایی از جدول t است که توسط پرس-وجوی q_i مورد دستیابی قرار گرفته است. پارامتر $Count_{q_i}$ تعداد تکرار پرس‌وجو q_i در بارکاری برنامه است.

معیار TAF برای کل شِمای پایگاه داده برابر با رابطه (۲) است:

$$TAF(schema) = \sum_{j=1}^{m} \mathrm{AF}(t_j) \qquad (۲)$$

m تعداد جدول‌های شِمای طراحی شده است. هر چه معیار TAF از شِمای پایگاه داده کمتر باشد، شِما بهینه‌تر است. روش پیشنهاد شده در این تحقیق با کمینه کردن تعداد دسترسی به پایگاه داده منجر به کاهش چمشگیر معیار TAF و در نتیجه بهبود کارایی پرس‌وجوها (زمان پردازش) می‌شود.

افزونی داده (DR): غیرنرمالسازی شِمای پایگاه داده منجر به افزونی داده و کپی های زائد از ویژگی‌های موجودیت‌ها می‌شود. افزونی داده ارتباط مستقیم با سربار ذخیره‌سازی دارد و منجر به افزایش آن می‌شود. معیار افزونی داده می‌تواند توسط اندازه-گیری متوسط حجم داده هر ویژگی محاسبه شود (ستونی از جداولی که ویژگی در آن ظاهر شده است). این فاکتور نشان دهنده رشد حجم داده از شِمای طراحی شده در مقایسه با ذخیره ساده در شِمای نرمال شده است. هدف کاهش این معیار برای شِمای طراحی شده در حد امکان است. در این تحقیق با تعریف قوانین ادغام و نرمالسازی شِما برای پرس‌وجوهای نوشتن پرتکرار توانسته‌ایم این معیار را تا حدممکن کاهش دهیم.

معیار DR برای یک ویژگی $attr$ از موجودیت e به صورت رابطه (۳) تعریف می‌شود:

$$DR(e.attr) = \sum_{i=1}^{n} \mathrm{V}(t_i \,.\, e.attr)$$

n تعداد جداولی است که ویژگی $attr$ در آن ظاهر شده است و $\mathrm{V}(t_i \,.\, e.attr)$ متوسط سایز داده از ستونی در جدول t_i است که ویژگی $attr$ در آن ظاهر شده است.

معیار DR برای کل شِمای پایگاه داده برابر است با رابطه (۴):

$$TDR(schema) = \sum_{j=1}^{m} \mathrm{DR}(e.attr_j) \qquad (۴)$$

با توجه به شرح ارائه شده در بالا، ابتدا در این بخش دو آزمایش مختلف برای بررسی کیفیت شِمای پیشنهادی انجام شده است. در اولین آزمایش متوسط زمان اجرا برای تراکنش‌های توزیع-های مختلف RUBiS (جدول۲) نشان داده شده است. آزمایش دوم به محاسبه دو معیار AF و DR برای شِمای پیشنهادی می-پردازد. همچنین در آزمایشات، کیفیت شِمای پیشنهادی با شِمای نرمال شده و شِمای ایجاد شده توسط ابزار NoSE مقایسه شده است. سپس در آزمایشی دیگر به بررسی کارایی الگوریتم تولید شِمای پیشنهادی پرداخته‌ایم.

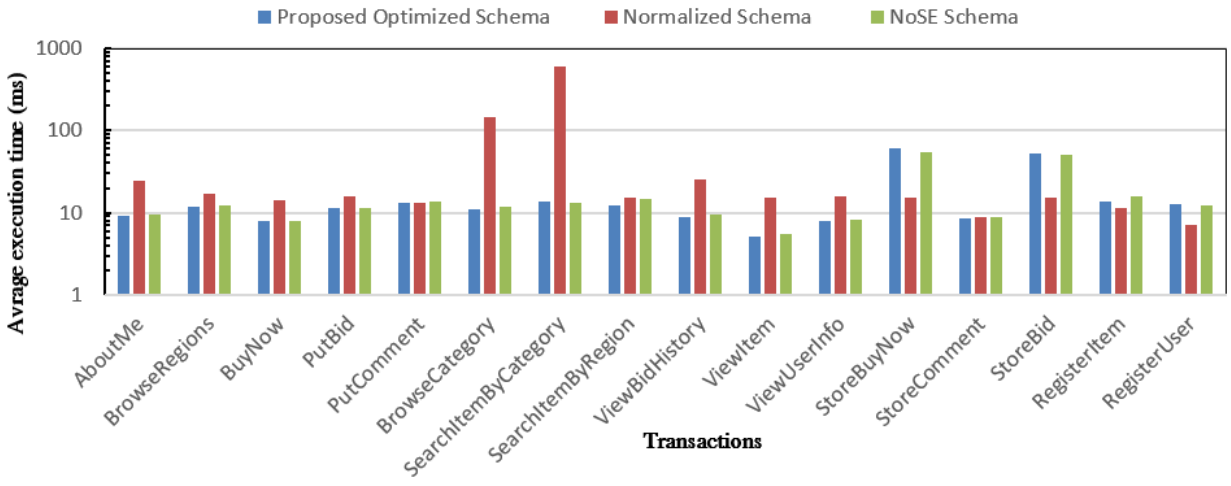
شکل ۳ و ۴ نتایج اولین آزمایش را نشان می‌دهند. نتایج آزمایشات در شکل ۳ نشان می‌دهد که متوسط زمان اجرا برای تراکنش‌های فقط خواندی پر تکرار با استفاده از شِمای بهینه-یافته پیشنهادی بهتر از شِمای نرمال است. زیرا روش پیشنهادی ما با تحلیل اطلاعات بارکاری و استفاده از روش غیرنرمالسازی منجر به کارایی خوبی از پرس‌وجوهای خواندن می‌شود. در مقابل، زمان اجرا برای تراکنش‌های نوشتنی پرتکرار با استفاده از شِمای پیشنهادی عملکرد بهتری از شِمای نرمال ندارد. زیرا شِمای نرمال دانشی از وزن تراکنش‌ها در بارکاری ورودی ندارد و با سطح نرمالسازی بالایی ایجاد شده است.

متوسط زمان اجرای وزن‌دهی شده برای سه توزیع مختلف بارکاری، به ترتیب افزایش شدت نوشتن، در شکل ۴ نشان داده شده است. با افزایش شدت نوشتن در بارکاری، شِمای پیشنهادی نرمال‌تر شده و کارایی بارکاری به نتایج بدست آمده از شِمای نرمال شده نزدیک می‌شود. به طور کلی، شِمای پیشنهادی ما با تحلیل بارکاری، تعداد درخواست‌ها به پایگاه داده را کمینه کرده و بنابراین عملکرد بهتری از شِمای نرمال شده دارد. همچنین شِمای پیشنهادی با تخصیص وزن به هر پرس‌وجو و تعیین اولویت طراحی برای پرس‌وجوهای بارکاری، تعداد درخواست‌ها به پایگاه داده را برای کل بارکاری برنامه کمینه کرده و بنابراین عملکرد بهتری از شِمای NoSE دارد. زیرا اگر اولویت با پرس‌وجوی خواندن است، روش پیشنهادی ما برای هر پرس‌وجوی خواندن یک جدول ایجاد می‌کند که به طور مستقیم و فقط با ارسال یک درخواست به پایگاه داده به آن پرس‌وجو پاسخ می‌دهد. در غیر اینصورت، الویت با پرس-وجوی نوشتن است و روش پیشنهادی با استفاده از نرمالسازی، سرعت اجرای پرس‌وجوی نوشتن را افزایش می‌دهد. در بارکاری light bidding که الویت فقط با پرس‌وجوهای خواندن است، متوسط زمان اجرای کل بارکاری از شِمای پیشنهادی تقریبا با شِمای NoSE یکسان است. اما در بارکاری bidding و heavy bidding که الویت می‌تواند با پرس‌وجوی نوشتن باشد، روش پیشنهادی از تکنیک نرمالسازی استفاده می‌نماید و در نتیجه عملکرد بهتری از NoSE دارد.

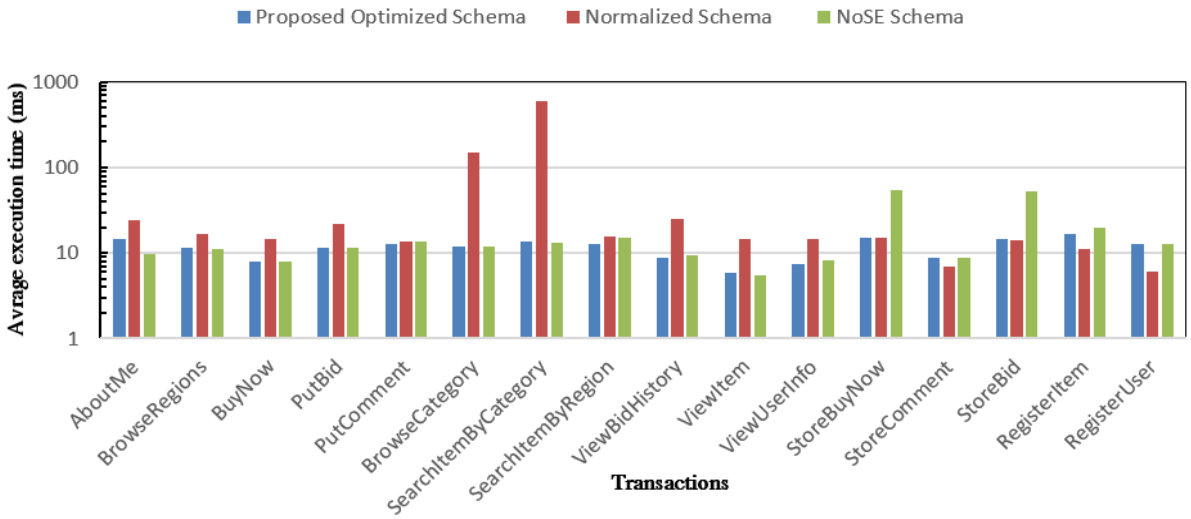
برای شفاف سازی نتایج بدست آمده در شکل ۴، تعداد درخواست‌های ارسالی به پایگاه داده را با در نظر گرفتن اثر وزن هر تراکنش در جدول ۳ برای شِمای پیشنهادی و شِمای NoSE نشان داده شده است. همانطور که نتایج نشان می‌دهد، شِمای پیشنهادی در مقایسه با شِمای NoSE تعداد درخواست‌های ارسالی به پایگاه داده برای پاسخ به پرس‌وجوها را به طور قابل ملاحظه‌ای کاهش می‌دهد و بنابراین عملکرد بهتری دارد و بهینه‌تر است.

در آزمایش دوم به محاسبه دو معیار AF و DR پرداخته شده است. مقادیر بدست آمده از این دو معیار برای سه شِمای بهینه‌یافته پیشنهادی، نرمال شده و NoSE در جدول ۴ نشان داده شده است. ردیف اول از جدول ۴ نشان می‌دهد که معیار TAF از شِمای پیشنهادی به طور چشمگیری در مقایسه با شِمای نرمال کاهش یافته است. این کاهش به سبب کمینه کردن تعداد درخواست‌ها به پایگاه داده است. با افزایش شدت نوشتن در بارکاری، شِمای بهینه‌یافته پیشنهادی نرمال‌تر شده و مقدار TAF به نتایج بدست آمده از شِمای نرمال نزدیک‌تر می-شود. مقدار معیار TDR در ردیف آخر جدول ۴ نشان می‌دهد که رشد حجم داده از شِمای پیشنهادی با افزایش شدت نوشتن به نتایج شِمای نرمال نزدیکتر می‌شود. غیرنرمالسازی شِمای پایگاه داده برای بهینه سازی شِما در روش پیشنهادی منجر به افزونی داده و افزایش سربار ذخیره‌سازی در مقاسه با شِمای نرمال شده می‌شود. هدف کاهش این معیار برای شِمای طراحی شده در حد امکان است. همانطور که در جدول ۴ مشاهده می-شود، در این تحقیق با تعریف قوانین ادغام و نرمالسازی شِما برای پرس‌وجوهای نوشتن پرتکرار توانسته‌ایم این معیار را تا حدممکن کاهش دهیم. همچنین نتایج آزمایشات در این جدول نشان می‌دهد که شِمای پیشنهادی در مقایسه با شِمای NoSE منجر به مقادیر کمتری برای دو معیار TAF و TDR می‌شود. در نتیجه روش پیشنهادی شِمای بهینه‌تری را در مقایسه با ابزار NoSE تولید می‌کند، در حالیکه منجر به سربار ذخیره‌سازی کمتری می‌شود.

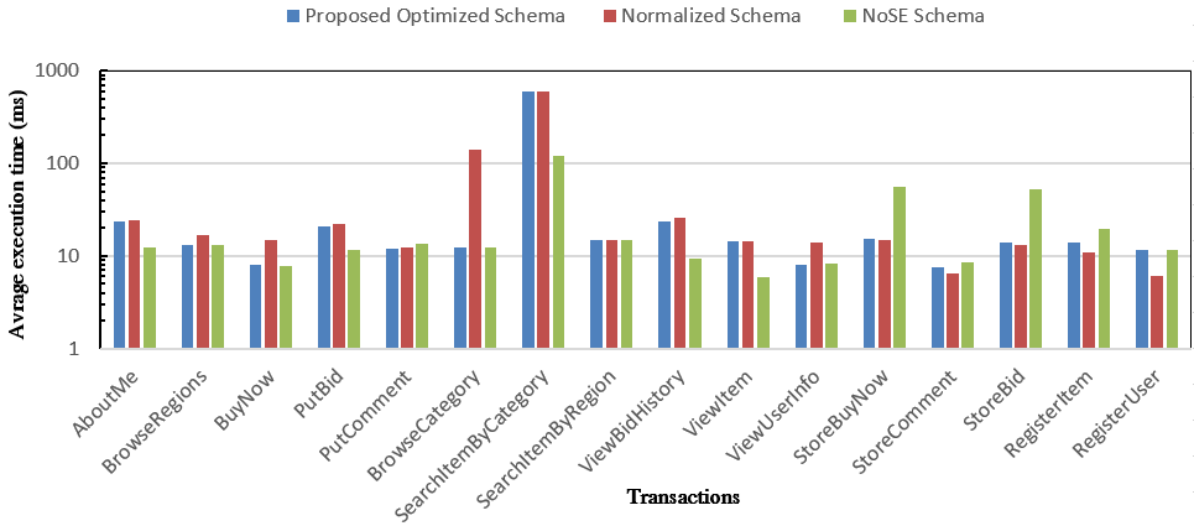
برای ارزیابی کارایی و پیچیدگی الگوریتم پیشنهاد شِما، باید با افزایش اندازه بارکاری (افزایش تعداد پرس‌وجوها و بروزرسانی-ها) زمان اجرای روش برای تولید شِما را اندازه‌گیری و ارزیابی کرد. بنابراین آزمایش سوم زمان تولید شِما پیشنهادی با تغییر سایز بارکاری پایگاه داده را نشان می‌دهد. برای ارزیابی زمان تولید شِما با بارکاری بزرگتر از RUBiS، به روشی مشابه با [۱۸] عمل کرده‌ایم و در نتیجه توانسته‌ایم مدل مفهومی و پرس-وجوهایی بر روی این مدل به طور تصادفی برای ورودی روش پیشنهادی ایجاد کنیم. شکل ۵ نتایج آزمایشی را نشان می‌هد که در آن با بارکاری مشابه با بارکاری RUBiS شروع شده است و سپس سایز بارکاری توسط ضرب کردن تعداد موجودیت‌ها در



(a)

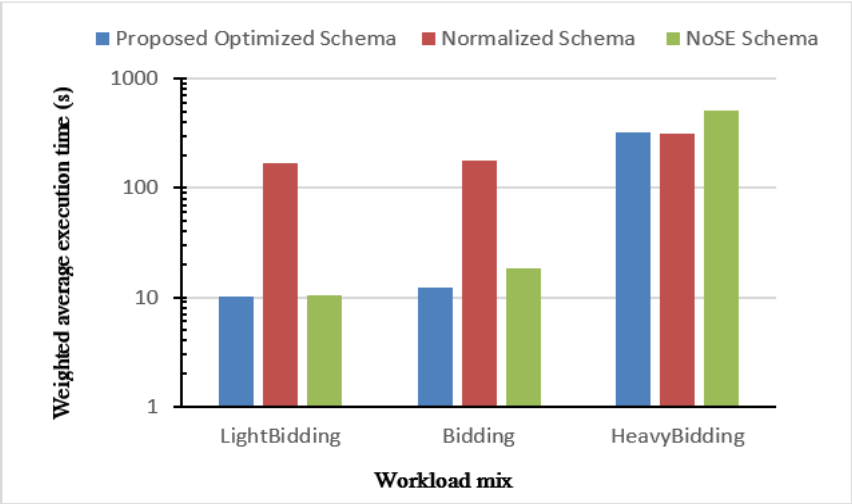


(b)



(c)

شکل ۳: متوسط زمان اجرا برای انواع تراکنش‌ها در ترکیب‌های مختلف بارکاری RUBiS. (a) Light bidding (b) Bidding (c) Heavy Bidding



شکل ۴: متوسط زمان اجرای وزن‌دهی شده برای سه توزیع مختلف بارکاری RUBIS

جدول ۳: تعداد درخواست‌ها برای هر تراکنش در بارکاری bidding

Transaction	The number of requests * Count	
	NoSE Schema	Proposed Schema
BrowseCategories	2 * 6617	2 * 6617
ViewBidHistory	2 * 1335	2 * 1335
ViewItem	2 * 12265	2 * 12265
SearchItemsByCategory	1 * 13790	1 * 13790
ViewUserInfo	2 * 2150	2 * 2150
RegisterItem	9 * 462	8 * 462
RegisterUser	2 * 929	2 * 929
BuyNow	2 * 1005	2 * 1005
StoreBuyNow	17 * 956	11 * 956
PutBid	3 * 4670	3 * 4670
StoreBid	17 * 4670	9 * 4670
PutComment	3 * 402	3 * 402
StoreComment	5 * 389	5 * 389
AboutMe	7 * 1477	9 * 1477
SearchItemsByRegion	1 * 5489	1 * 5489
BrowseRegions	1 * 1963	1 * 1963
Total	**183,134**	**156,540**

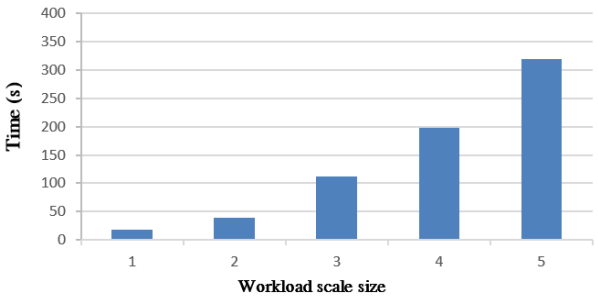
جدول ۴: تعداد درخواست‌ها برای مقادیر بدست آمده برای دو معیار TAF **و** TDR

پارامتر	**Heavy Bidding**			**Bidding**			**Light Bidding**		
	شمای نرمال-شده	شمای بهینه-یافته پیشنهادی	شمای NoSE	شمای نرمال-شده	شمای بهینه-یافته پیشنهادی	شمای NoSE	شمای نرمال-شده	شمای بهینه-یافته پیشنهادی	شمای NoSE
تکرار دسترسی کل (TAF)	18,955,583	18,399,164	19,721,438	13,176,304	4,016,446	5,264,351	13,006,518	3,761,256	3,792,521
افزونی داده کل (TDR)	6,770,486	7,490,324	8,150,324	6,770,486	8,160,324	12,480,324	6,770,486	15,510,324	16,210,324

مدل مفهومی و تعداد جملات پرس‌وجو در یک ضریب ثابت افزایش یافته است. لازم به ذکر است که سایز بارکاری (تعداد پرس‌وجوهای خواندن و نوشتن) مستقل از وزن هر پرس‌وجو است. شکل ۵ نشان می‌دهد که زمان تولید شِما با افزایش سایز بارکاری به طور خطی افزایش می‌یابد. این رابطه خطی به علت مراحل اکتشافی روش پیشنهادی است که منجر به کاهش تعداد جداول می‌شود و همچنین نشان دهنده مقیاس‌پذیری روش‌پیشنهادی است. در این آزمایش، از ضریب همبستگی(r) برای اثبات رابطه خطی بین دو متغییر زمان تولید شِما و سایز بارکاری در نمودار مربوطه استفاده شده است. ضریب همبستگی شاخص توصیفی است که وجود رابطه خطی بین دو متغیر را سنجش می‌کند. از آنجایی که این دو متغییر در مقیاس یکسانی نیستند، ابتدا با روش نرمال‌سای max-min مقادیر دو متغییر به بازه [0.0,1.0] نگاشت شده است و سپس با استفاده از فرمول (۵) مقدار r محاسبه شده است.

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i-\tilde{x}}{s_x}\right)\left(\frac{y_i-\tilde{y}}{s_y}\right) \qquad (۵)$$

متغییر x_i زمان نرمال شده و y_i سایز بارکاری نرمال شده است. مقدار همبستگی محاسبه شده از فرمول (۵) برابر ۰.۹۷ است که به عدد یک بسیار نزدک است و نشان دهنده رابطه خطی قوی بین این دو متغییر است. بنابراین پیچیدگی زمانی روش پیشنهادی ما با رشد بارکاری ورودی (n) رابطه خطی دارد و برابر با $O(n)$ است. در مقابل، همانطور که در [۱۸] نشان داده شده است، زمان اجرای ابزار تولید شِمای NoSE بطور نمایی با سایز بارکاری افزایش می‌یابد.



شکل ۵: زمان تولید شِما برای سایزهای بارکاری مختلف

۶- نتیجه‌گیری

پایگاه داده‌های NoSQL برای ذخیره سازی داده‌ها در مدل داده بدون شِما یا شِمای با انعطاف پذیری بالا توسعه داده شده‌اند. مساله طراحی شِما در پایگاه داده NoSQL متفاوت از پایگاه داده رابطه‌ای می‌باشد و در مقایسه با آن با چالش‌های جدیدی روبرو است. در این مقاله یک روش طراحی شِما خودکار بر مبنای مدل مفهومی و بارکاری برنامه در پایگاه داده ستون‌گرا NoSQL ارائه شد. این روش با کمینه کردن تعداد درخواست‌ها به پایگاه داده منجر به طراحی شِمای بهینه‌یافته برای کل بارکاری برنامه می‌شود، به طوریکه سربار ذخیره سازی و سربار اجرای پرس‌وجوهای نوشتن را کاهش می‌دهد. برای رسیدن به این هدف، روش پیشنهادی با در نظر گرفتن اطلاعات مربوط به وزن هر پرس‌وجو در بارکاری و تعیین الویت طراحی شِما برای پرس‌وجوهای بارکاری توانست توازن بین نرمالسازی و غیرنرمالسازی در طراحی شِما را کنترل کند. نتایج حاصل شده از آزمایش‌ها نشان داد که شِما خودکار تولید شده از روش پیشنهادی منجر به کارایی خوب بارکاری می‌گردد.

از کارهای آینده، توسعه روشی مشابه برای خودکارسازی شِما در انواع دیگری از پایگاه داده NoSQL نظیر پایگاه داده سندگرا و کلید/مقدار می‌باشد.

مراجع

[1] S. Chaudhuri and V. Narasayya, “Self-tuning database systems: a decade of progress”, in Proceedings of the 33rd international conference on Very large data bases, pp. 3-14, 2007.

[2] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala, "Database tuning advisor for microsoft SQL server 2005: demo," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 930-932: ACM.

[3] N. Bruno and S. Chaudhuri, "An online approach to physical design tuning," in *IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007, pp. 826-835: IEEE.

[4] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin, "Automatic SQL tuning in Oracle 10g," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, Toronto, Canada, 2004, pp. 1098-1109: VLDB Endowment.

[5] S. Papadomanolakis and A. Ailamaki, "Autopart: Automating schema design for large scientific databases using data partitioning," in *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, 2004, pp. 383-392: IEEE.

[6] K.-U. Sattler, I. Geist, and E. Schallehn, "Quiet: Continuous query-driven index tuning," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*, 2003, pp. 1129-1132: VLDB Endowment.

[7] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "Colt: continuous on-line tuning," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 793-795: ACM, 2006.

[8] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley, "DB2 advisor: An optimizer smart enough to recommend its own indexes," in *Proceedings. 16th International Conference on Data Engineering*, pp. 101-110: IEEE. 2000.

[9] D. C. Zilio *et al.*, "DB2 design advisor: integrated automatic physical database design," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 1087-1097, 2004.

[10] A. Rasin and S. Zdonik, "An automatic physical design tool for clustered column-stores," in *Proceedings of the 16th*

International Conference on Extending Database Technology, 2013, pp. 203-214: ACM.

[11] E. Hewitt, *Cassandra: the definitive guide*, O'Reilly Media, 2010.

[12] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," ACM SIGOPS Operating Systems Review, vol. 44, no. 2, 2010.

[13] M. Stonebraker *et al.*, "C-store: a column-oriented DBMS," in *Proceedings of the 31st international conference on Very large data bases*, pp. 553-564, 2005.

[14] A. Pavlo *et al.*, "Self-Driving Database Management Systems" , in CIDR 2017, Conference on Innovative Data Systems Research, January 8-11, Chaminade, CA**,** 2017.

[15] D. Bermbach, S. Müller, J. Eberhardt, and S. Tai, "Informed schema design for column store-based database services," in *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 163-172: IEEE, 2015.

[16] A. Chebotko, A. Kashlev, and S. Lu, "A big data modeling methodology for Apache Cassandra," in *2015 IEEE International Congress on Big Data*, pp. 238-245: IEEE, 2015.

[17] C. de Lima and R. dos Santos Mello, "A workload-driven logical design approach for NoSQL document databases," in *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, p. 73: ACM, 2015.

[18] M. J. Mior, K. Salem, A. Aboulnaga, and R. Liu, "NoSE: Schema design for NoSQL applications," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[19] Vajk, L. Deák, K. Fekete, and G. Mezei, "Automatic NoSQL schema development: A case study," in *Artificial Intelligence and Applications*, pp. 656-663, 2013.

[20] T. Vajk, P. Feher, K. Fekete, and H. Charaf, "Denormalizing data into schema-free databases," in *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pp. 747-752: IEEE, 2013.

[21] M. Boussahoua, O. Boussaid, and F. Bentayeb, "Logical Schema for Data Warehouse on Column-Oriented NoSQL Databases", in International Conference on Database and Expert Systems Applications, pp. 247-256, 2017.

[22] F. Yang, D. Milosevic, and J. Cao, "Optimising column family for OLAP queries in HBase," *International Journal of Big Data Intelligence*, vol. 4, no. 1, pp. 23-35, 2017.

[23] A. A. Imam, S. Basri, R. Ahmad, J. Watada, and M. T. González-Aparicio, "Automatic schema suggestion model for NoSQL document-stores databases," Journal of Big Data, vol. 5, no. 1, p. 46, 2018.

[24] F. Abdelhedi, R. Jemmali, and G. Zurfluh,, "Relational Databases Ingestion into a NoSQL Data Warehouse". *arXiv preprint arXiv:2203.06949*. 2022

[25] I.B. Messaoud, A.A Alshdadi and J. Feki, "Building a Document-Oriented Warehouse Using NoSQL". *International Journal of Operations Research and Information Systems (IJORIS)*, *12*(2), pp.33-54. 2021

[26] F. Abdelhedi, A.A. Brahim, H. Rajhi, R.T. Ferhat, and G. Zurfluh, "Automatic Extraction of a Document-oriented NoSQL Schema", In Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1, pages 192-199 , 2021

[27] G. Daniel, A. Gómez and J. Cabot," UMLto [No] SQL: Mapping Conceptual Schemas to Heterogeneous Datastores". In *2020 13th International Conference on Research Challenges in Information Science (RCIS)* (pp. 1-13). IEEE. 2020.

[28] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world". *Computer Standards & Interfaces*, *67*, p.103149. 2020

[29] P. Atzeni, F. Bugiotti, and L. Rossi, "Uniform access to NoSQL systems," Information Systems, vol. 43, pp. 117-133, 2014.

[30] V. Noguera and D. Lucrédio, "Implementing a Classic ER Algebra to Automatically Generate Complex Queries for Document-Oriented Databases," in Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse, pp. 43-52: ACM, 2019.

[31] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data ",ACM Transactions on Database Systems (TODS), vol. 1, no. 1, pp. 9-36, 1976.

پاورقی‌ها:

[1] Self-adaptive

[2] Autonomic Database

[3] Self-Tuning

[4] Automated Schema Database Tuning

[5] Automated Physical Database Design

[6] Workload

[7] Schema-less

[8] Wide Column Store

[9] Column Families

[10] Hbase

[11] Fault tolerance

[12] Linear Scalable

[13] High Availability

[14] Data center

[15] Cassandra Query Language

[16] Client-Side Join

[17] Advisory Tools

[18] Self-driving

[19] binary integer program

[20] Relative Frequency

[21] RUBIS: Rice University bidding system. https://projects.ow2.org/view/rubis/

[22] Average Execution Time

[23] Time Schema Generation

[24] Average Execution Time

[25] Time Schema Generation