筑波大学 情報学群 情報メディア創成学類

卒業研究論文

NoSQL型データベースシステムでの実体化ビュー選択に関する研究

高木 颯汰

指導教員　古瀬 一隆 陳 漢雄

2019年1月

概要

本論文では NoSQL の一種であるドキュメント指向データベースに実体化ビューを導入する事によって問い合わせ処理を高速化する手法を提案する．ドキュメント指向データベースでは従来のリレーショナルデータベースにあったような参照型のデータ構造に加えて埋込型のデータ構造を選択できる．参照先の内容を埋め込む事によって結合処理をしなくて済むが，ファイルサイズが大きくなる傾向にあり，フラグメンテーションが発生し，逆にパフォーマンスが落ちる可能性がある．そこで本手法ではリレーショナルデータベースで実現されている実体化ビューの概念を NoSQL にも応用する事で，問い合わせ処理を自動的に高速化する．具体的には，頻繁に問い合わせのある結合処理や集計処理を自動的に検知してその部分のみ予め実体化することでデータベースアクセスの高速化を実現している．実体化する箇所の選択を自動化することにより，データベースシステム管理者が行なっていた作業を簡略化し，客観的で正確な実体化ビュー選択が可能となる．

目 次

図 目 次

表 目 次

第1章　はじめに

数年前までは主要なデータストアとして，リレーショナルデータベースがあげられることがほとんどであった．それは多くの開発者が SQL に慣れ親しんでおり，正規化されたデータモデル，トランザクションの必要性，耐久性のあるストレージエンジンが提供する保証を受けられるからである [1]．しかし近年高いスケーラビリティや大量なデータ処理が得意であることなどから NoSQL に対する需要が急激に増えている．

例えば NoSQL の一種のドキュメント指向データベースはデータベースの構造を表すスキーマを定義する必要がなく，大量なデータを事前準備なしで格納することができる．従来のリレーショナルデータベースにあったような参照型のデータ構造に加えて埋込型のデータ構造を選択できる．型宣言の必要のないスクリプト言語と相性が良いことなども合間って，プロトタイプを高速に開発することが求められるビジネスの現場で採用されることが増えている [2].

一方でドキュメント指向データベースの特徴とも言える階層的なデータモデルが更新処理速度の低下やデータ参照の柔軟性を低下を招くことがある．これを防ぐためにはドキュメントに階層的に埋め込むフィールドを適切に選択する必要がある．本論文ではこの選択の自動化し，ドキュメント指向データベースのデータモデルのチューニングを行い，データアクセスを高速化する．

本論文の構成は以下の通りである．まず，第 2 章において関連研究について紹介する．次に，第 3 章において本研究の提案手法について説明をし，第 4 章にて提案手法に関する実験を行い，提案手法の有効性を確める．第 5 章において実験の結果と考察を述べ，最後に第 6 章において本論文のまとめと今後の課題を示す．

第2章　関連技術

2.1　Materialized View

リレーショナルデータベースにおけるビュー (view) はリレーショナルデータモデルの発案者であるコッドにより導入された概念であり [3]，1 つ以上の表（または他のビュー）から任意のデータを選択し，それらを表したものである．ビューの実体はデータを持たない SQL 文であり，実行された際にはバックグラウンドで SELECT 処理が毎回実行される．それに対して実体化ビュー（Materialized View）はビューと同じく複数の表の結合処理や集計処理を行うが，その結果を実際のテーブルに保持する．保持された実体化ビューは元のテーブルが更新されるたびに更新される．そのため，最新でない状態を取得する可能性はあるが，結合処理が必要ないため効率的なアクセスが可能になる．その一方，更新処理が増加するので実体化ビュー化する部分の選択は慎重に行う必要があり，この作業を自動化する研究が行われている [4]．図 2.1 は 1 対 1 のデータモデルの実体化ビューを図示したものである．

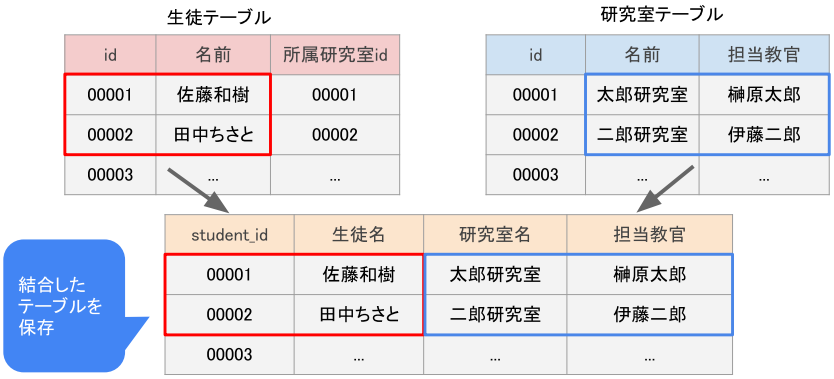

図 2.1: 実体化ビュー

多対多の結合を表す際には中間テーブルを用意してそれぞれのテーブルからレコードを結合する．その際の流れを図 2.2 に示す．履修中間テーブルに生徒 id と授業 id を格納している．id が 00001 の生徒が履修している授業を取得する際には履修中間テーブルの student_id が 00001 のレコードを取得し，付随する class_id を用いて授業の情報を取得する．結合元のテーブルのレコード数が無数にある場合には結合元テーブルでの検索時間が増加し，結合元の

テーブルのレコード数の増加と共に中間テーブルのレコード数も増える傾向にあるので，中間テーブルでの検索時間も増加する．Materialized View を用いた際のメリットとして，結合

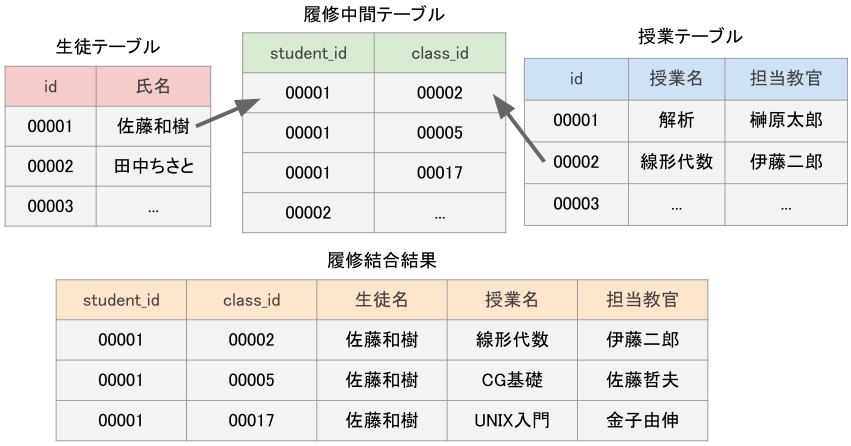

図 2.2: 多対多の結合

処理や集計処理が不要になり高速化することに加え，問い合わせに用いる SQL 文が簡素化することが挙げられる．表 2.1 は図 2.2 の生徒テーブルを student，授業テーブルを class，履修中間テーブルを course_selection，生徒の履修結合結果を得るために作成した Materialized View を mv_course_selection とした際に，id が 00001 の生徒が履修している授業を取得する SQL 文を Materialized View を使用する場合としない場合を比較したものである．

表 2.1: 生徒テーブルと授業テーブルを結合する SQL

Materialized View を使用しない	SELECT student.id AS student_id, class.id AS class_id, student.name AS student_name, class.name AS class_name, class.teacher AS teacher FROM student, class, course_selection WHERE course_selection.student_id = 00001 AND course_selection.student_id = student.id AND course_selection.class_id = class.id;
Materialized View を使用する	SELECT student_id, class_id, student_name, class_name, teacher FROM mv_course_selection WHERE student_id = 00001;

2.2 NoSQL

NoSQL とは，“Not only SQL”の略称であり，SQL を用いないデータベースの総称を表す [5]．情報の大規模化が進み，ビッグデータと呼ばれる概念が登場すると共に，構造が複雑な様々なデータが登場するようになった．NoSQL は，そのような複雑な構造のデータに柔軟に対応し処理を行うことができる．Google や Amazon，Twitter など，世界的規模を誇る企業が NoSQL データベースを利用しており，今後ますますデータの大規模化が進む現代社会において，重要な役割を果たすデータベースである [5]．NoSQL データベースはキー・バリュー型，カラム指向型，ドキュメント指向型，グラフ型の 4 種類の型に大別することができる．キー・バリュー型は，インデックスであるキーと値であるバリューのペアでデータが構成され，キーを指定することでデータを呼び出すことができる．カラム指向型は行に対してキーが付され，それが複数の列 (カラム) に対応する形のデータモデルである．ドキュメント指向型は，JSON や XML などの形式で記述されたドキュメントの形でデータを扱うデータモデルである．グラフ型は，データ間の関係性をグラフの構造で表すデータモデルである [5]．

2.3 MongoDB

MongoDB とは，JSON や XML などの形式で記述されたドキュメント指向型のデータを扱う NoSQL データベースの代表的なものの一つである．RDB とは違い，スキーマの定義を必要としない [5][6]．また，JSON 形式のデータを扱うため，Web システムなどに利用しやすい．MongoDB においては，RDB のテーブルにあたるものとしてコレクション，RDB の行にあたるものとしてドキュメント，RDB の列にあたるものとしてフィールドというデータ構想が使われる．

MongoDB ではデータの格納に JSON をバイナリエンコーディングした BSON 形式を用いる [7]．RDBMS と MongoDB における一般的な検索のクエリを表 2.2 に，更新のクエリを表 2.3 に示す．また，MongoDB はクエリの結果を JSON 形式で返す．返却されるクエリセットの例を図 2.3 に示す．

表 2.2: RDBMS と MongoDB における検索クエリ

RDBMS	SELECT * FROM comments WHERE story = ”Next Generations”;
MongoDB	db.comments.find({story: ”Next Generations”});

表 2.3: RDBMS と MongoDB における更新クエリ

RDBMS	UPDATE comments SET story = ”Next Generations 2” WHERE story = ”Next Generations”;
MongoDB	db.comments.update({story: ”Next Generations”}, {$set: {story: ”Next Generations 2”}});

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "speak": {
    "speaker": "名無しさん",
    "comment": "この話はとても面白い."
  },
  "story": "Next Generations",
  "created_at": "2015-04-23 03:07:27",
  "updated_at": "2015-04-23 03:07:27"
}
```

図 2.3: MongoDB から返されるクエリセットの例

2.4 Restful API

REST とは Roy Fielding が提唱した概念であり [8],“REpresentational State Transfer”の略である．分散システムにおける複数のソフトウェアを連携させるのに適した考え方であり，やりとりされる情報はそれ自体で完結して解釈できるステートレス性，全てのリソースが一意的なアドレスを持つアドレス可能性，他の基盤的な機能を用いずに別の情報や状態を含むことで他のリソースを参照できる持続性，HTTP メソッド（“GET”や“POST”など）の統一インターフェースを提供していることなどの原則から成る．REST の原則に則り構築された HTTP の呼び出しインターフェースを RESTful API と呼ぶ．本論文では RESTful API をミドルウェアに実装し実験を行う．

第3章　提案手法

3.1　ドキュメント指向型データベースにおける実体化について

ドキュメント指向型データベースの特徴として埋め込み（embed）がある．従来の RDB では複数の表による 1 対多や多対多の関係を表す際に，参照先のプライマリーキーのみを保存して SELECT される際に結合処理を行う．それに対してドキュメント指向型データベースでは参照先の実データを参照元に埋め込むことができ，これによって結合処理を省くことができる．埋め込み先が複数の場合には更新処理が増加し，従来の参照型に比べてデータアクセスの柔軟性が損なわれるというデメリットがある [1]. 図 3.1 はドキュメント指向型データベースの参照型を，図 3.2 は図 3.1 のデータを埋込型で表した図である．また．図 3.2 と図 3.1 のドキュメントを得るためのクエリを表 3.1 に示す．id が 123456 のドキュメントを更新するクエリを表 3.2 に示す．なお，表 3.2 の埋込型に関しては，埋め込まれているコレクション全てに対してクエリを実行する必要がある．

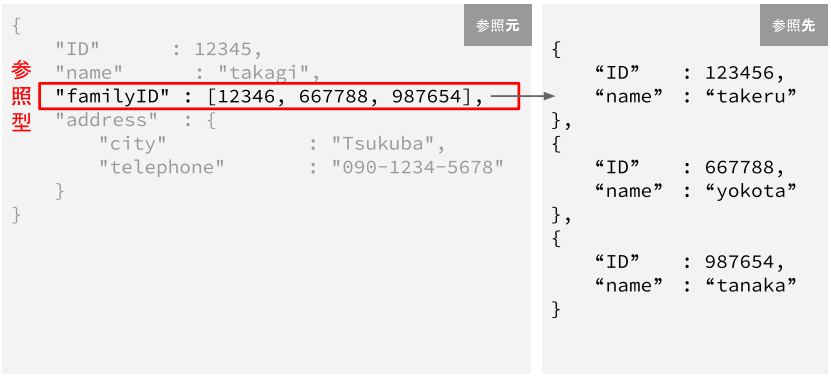

図 3.1: 参照型

表 3.1: MongoDB における参照型データモデルと埋込型データモデルでの検索クエリ

参照型	db.people.aggregate([{ $match: {ID: 12345}}, { $lookup: { from: ”families”, localField: ”familyID”, foreignField: ”ID”}}]);
埋込型	db.people.find({ID: 12345});

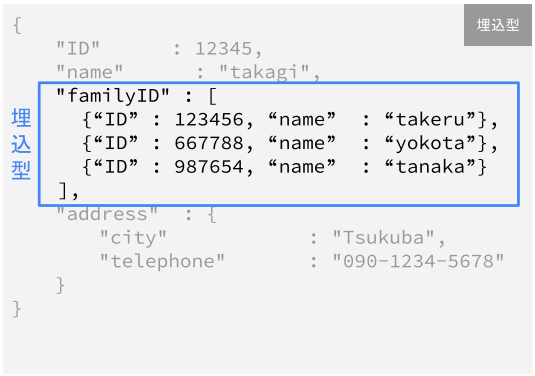

図 3.2: 埋込型

表 3.2: MongoDB における参照型データモデルと埋込型データモデルでの更新クエリ

参照型	db.family.update({ ID: 123456},{ $set: { name: ”takebayashi”}});
埋込型	db.people.update({”family.ID”: 12345}, { $set: {family.$.name: ”takebayashi”}});

全てのドキュメントを埋め込み型として保存すると埋め込み先のドキュメントの更新処理が増え，著しく更新時間が増加する為，データモデルとして最適とは言えない．埋め込み型のデータモデルとして保存するコレクションを最適に選択し，データモデルを最適化することがドキュメント指向型データベースを高速に使用することに繋がる．本論文ではこのコレクションの選択を自動化する．

3.2 提案手法の構成

本論文ではドキュメント指向型データベースの埋込型データモデルをリレーショナルデータベースの実体化ビューと置き換えて考える．ドキュメント指向型データベースのよくアクセスされる部分や処理速度がネックとなっている部分を埋込型として別コレクションに保持することで“実体化ビュー作成”，どの部分を埋込型にするかの判断を“実体化ビュー選択”とする．図 3.3 は本論文での“実体化ビュー作成”を示したものである．

実装システムについて図 3.4 に示す．ユーザーからのデータアクセスから実体化ビュー作成までの流れを図 3.4 を用いて説明する．図 3.4-①まずユーザーがアプリケーションからミドルウェアに対してデータアクセスの要求する．図 3.4-②ミドルウェアでは，頻繁にアクセスされるドキュメントを分析するために，クエリに関するログを残す．図 3.4-③次に MongoDB に対してクエリを発行する．図 3.4-④ MongoDB から返ってきたクエリセットをアプリケーションに返却する．図 3.4-⑤クエリログを解析し，ボトルネックとなっているところや呼び出し回数の多い条件の実体化ビューを作成する．図 3.4-⑥実体化したドキュメントに関してログに記録する．

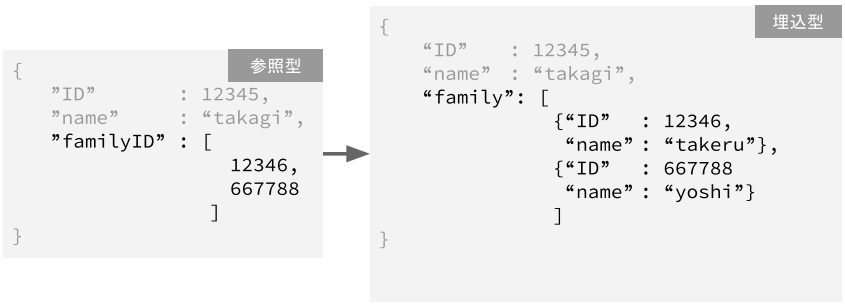

図 3.3: 参照型から埋込型への書き換え

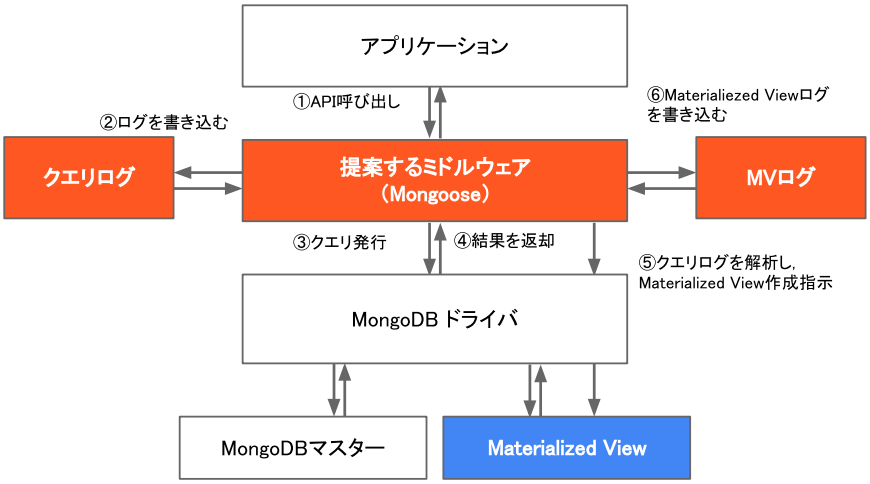

図 3.4: 提案ミドルウェア（実体化前）

実体化した後のデータアクセスの流れを図 3.5 に示す．図 3.5-①’アプリケーションからデータベースにアクセスがあった場合，まずログからアクセスされたデータが実体化されているか判定する．図 3.5-②’実体化されている場合はクエリを書き換えて実体化ビューから結果を取得する．図 3.5-③’アプリケーションに結果を返す際には元のクエリに合うように適宜変換する．

3.3 実体化アルゴリズムについて

クエリログから実体化するコレクションを決定する条件を適切に設定することで，実体化ビュー選択を自動化することができる．この実体化条件については以下の条件が考えられる．

1. クエリログの統計
2. ドキュメント内の参照数

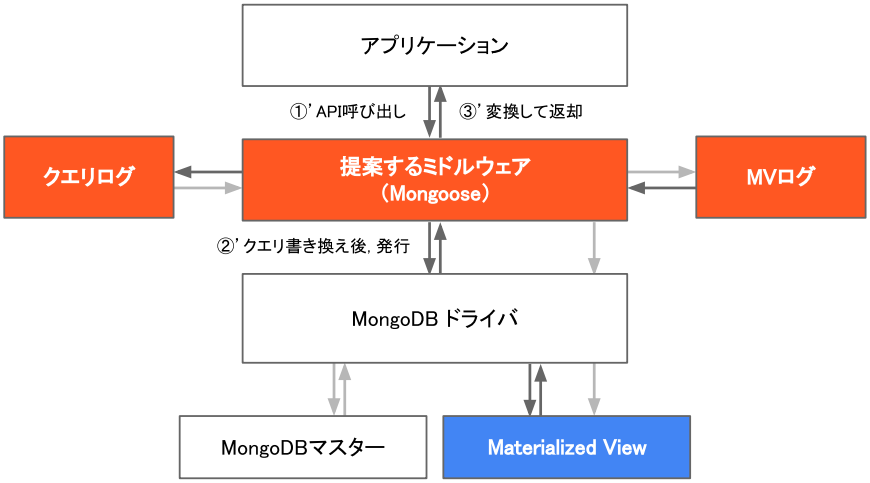

図 3.5: 提案ミドルウェア（実体化後）

1 の条件は実際のクエリの検索回数・更新回数やその処理時間を元に実体化するコレクションを選定する．2 の条件ではドキュメント内の他コレクションへの参照数を元に実体化後の更新処理の増加を予想し，実体化するコレクションを選定する．本論文では更新処理が用意であり，クエリに柔軟に対応できるクエリログの統計を元に実体化する条件を作成する．

3.4 逆実体化アルゴリズムについて

クエリのログを元に実体化した場合，クエリの傾向が変わることで実体化していない状態の方が望ましくなる可能性や，コレクションの特性によっては実体化によるメリットがデメリットより少ない可能性がある．そのような事を防ぐために，定期的に実体化したコレクションに対してもメンテナンスを行い，場合によっては実体化したコレクションをオリジナルのデータモデルに戻すことが必要である．本論文では実体化したコレクションのデータモデルを元に戻す事を逆実体化と定義する．この逆実体化に対しても実体化同様，適切な逆実体化条件を設定する必要がある．

本論文ではクエリのログの統計から実体化後の検索処理時間と更新処理時間を比較し，各コレクションに対して逆実体化の必要性を確認する．

第4章　実験

4.1　Mongooseについて

Mongooseとは MongoDB 用モデリングツールで，Node.js の非同期環境でうまく動作することを目的として設計されている．Mongoose を使用すれば，モデルを定義して操作することで，MongoDB のコレクション/ドキュメントを操作できる [9]．本論文では Mongoose を用いて MongoDB を操作するミドルウェアを実装する．

4.2　実験環境

実験環境に関する情報を表 4.1 に示す．

表 4.1: 実験環境

マシン	MacBook (Retina, 12-inch, 2017)
プロセッサ	1.2 GHz Intel Core m3
メモリ	8 GB 1867 MHz LPDDR3
データベースシステム	Mongodb version 3.1.10

4.3　実験方法

本論文の実験で用いたコレクションは person コレクション，story コレクション，comment コレクション，publisher コレクションである．コレクションの構造，コレクション同士の参照に関しては図 4.1，図 4.2 に示す．story コレクションには筆者として 1 つの person ドキュメントの id を格納する．この story ドキュメントが検索された際には筆者の id を perosn コレクションから検索し，結合して結果を返す．同じようにファンとして person ドキュメントの id を配列で格納することで複数の person ドキュメントを story ドキュメントに埋め込む．実際の実験データではファンとして 100 の person ドキュメントの id を埋め込む．その際，100 回結合処理を実行することになる．出版社として publisher ドキュメントの id も格納する． 実験において使用するデータは全て参照型のデータモデルで挿入する．実体化していない状態で検索された場合には結合処理を行い結果を返す．実験に用いるドキュメントは Python のラ

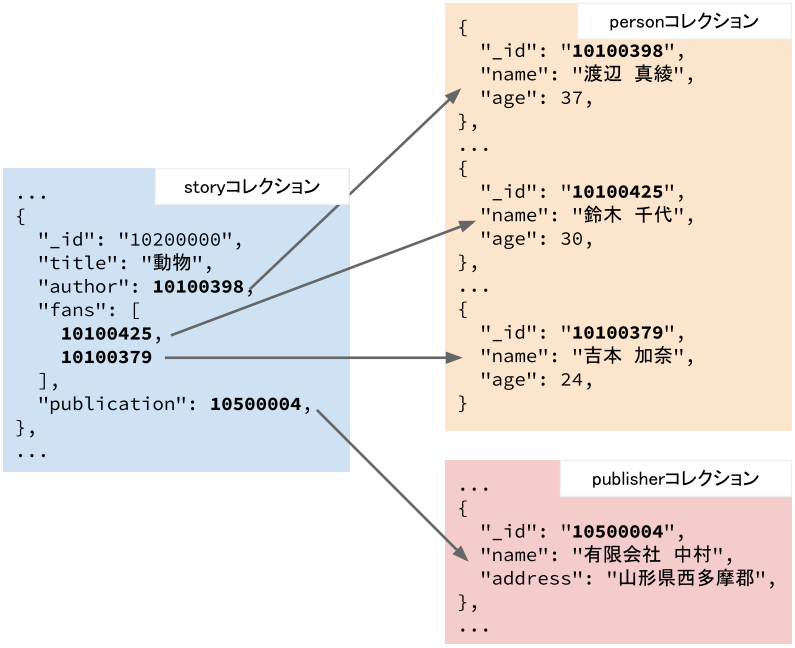

図 4.1: story コレクションから各コレクションへの参照

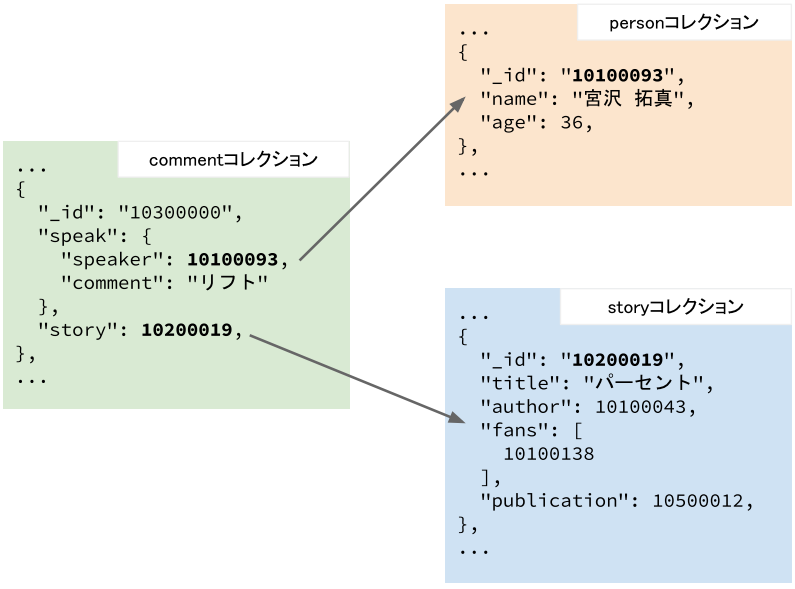

図 4.2: comment コレクションから各コレクションへの参照

イブラリである faker[10] を用いて作成した．各コレクションに対して 1000 ドキュメントを作成した．

実験では全てのコレクションが従来の参照型のデータモデルを用いたシステムと実体化条件を用いずに全てのコレクションを実体化したシステム，実体化条件を用いて適宜コレクションを実体化するシステムの 3 つのデータベースシステムを比較する．その際，実体化を用いたシステムでは検索や更新を提案ミドルウェアを用いて処理する．

この 3 つのシステムを比較する際，検索と更新の比率を変えた 4 つのクエリパターンを用いて行う．このクエリパターンを表に示す．このパターンを用いて実験 A D を行う．それぞ

表 4.2: クエリパターン

	検索回数	更新回数
パターン A	99	1
パターン B	97	3
パターン C	95	5
パターン D	90	10

れのパターンに対して，使用したクエリを表で示す．ここでは Mongoose の populate を用いて結合処理をおこなっている．また，提案手法を用いる場合には検索・更新に関わらずクエリが 5 回処理された際に実体化条件を検討し，適宜実体化ビューを作成，破棄を行う．

表 4.3: 実験で使用した検索クエリ

コレクション	クエリ
person	db.person.find({_id: testID})populate([]);
story	db.story.find({_id: testID}).populate([”author”, ”fans”, ”publication”, ”comments”]);
comment	db.comment.find({_id: testID}).populate([”speak.speaker”, ”story”]);
publisher	db.publisher.find({_id: testID}).populate([]);

表 4.4: 実験で使用した更新クエリ

コレクション	クエリ
person	db.person.update({_id: testID}, {$set: {name: ”太郎”}});
story	db.story.update({_id: testID}, {$set: {title: ”研修資料”}});
comment	db.comment.update({_id: testID}, {$set: {”speak.comment”: ”いい天気”}});
publisher	db.publisher.update({_id: testID}, {$set: {address: ”つくば市天王台”}});

第5章　結果・考察

5.1　実験Aの結果

実験結果を図に示す.

第6章　まとめ

謝辞

本研究を進めるにあたり，指導教員の古瀬一隆先生と陳漢雄先生から，丁寧かつ熱心なご指導を賜りました．ここに感謝の意を表します．また，研究室での議論を通じ，多くの知識をいただいた DSE 研究室の皆様に感謝いたします．

参考文献

[1] Kyle Banker. MongoDB イン・アクション. オライリージャパン, 第 1 版, 2012.

[2] 渡部徹太郎, 河村康爾, 北沢匠, 佐伯嘉康, 佐藤直生, 原沢滋, 平山毅, 李昌桓. RDB 技術者のための NoSQL ガイド. 秀和システム, 第 1 版, 2016.

[3] E. F. Codd. Recent investigations in relational data base systems. In *IFIP Congress*, 1974.

[4] Hoshi Mistry, Prasan Roy, S Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *ACM SIGMOD Record*, Vol. 30, pp. 307–318. ACM, 2001.

[5] 本橋信也, 河野達也, 鶴見利章. NOSQL の基礎知識 (ビッグデータを活かすデータベース技術). リックテレコム, 第 1 版, 2012.

[6] MongoDB. What is mongodb? https://www.mongodb.com/what-is-mongodb. (参照 2019-01-09).

[7] Guy Harrison. Next generation databases: Nosqland big data (english edition), 12 2015.

[8] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, Vol. 7. University of California, Irvine Irvine, USA, 2000.

[9] Valeri Karpov. Mongoose odm v5.4.4. https://mongoosejs.com/. (参照 2019-01-09).

[10] Flavio Curella. Welcome to faker's documentation! ¶ . https://faker.readthedocs.io/en/master/. (参照 2019-01-09).