

**אוניברסיטת בר אילן**



**תואר שני במנהל עסקים ומערכות מידע**

**פרקטיקום במערכות מידע**

**בנושא Whitepaper NoSQL**

**מגישים:**

**מנחה:**

תשפ"ה - 2024

תוכן עניינים

[**1.** תקציר מנהלים 4](#_Toc178276497)

[**2.** מבוא 5](#_Toc178276498)

[2.1. כללי**:** 5](#_Toc178276499)

[2.2. נתון מידע וידע: 6](#_Toc178276500)

[2.3. מסד נתונים: 7](#_Toc178276501)

[2.4. רקע והיסטוריה של מסדי נתונים: 7](#_Toc178276502)

[**3.** מסד נתונים רלציוני 9](#_Toc178276503)

[3.1. תיאור מסד הנתונים: 9](#_Toc178276504)

[3.2. שפת SQL: 11](#_Toc178276505)

[3.3. אלגברה רלציונית: 12](#_Toc178276506)

[3.4. נורמליזציה: 13](#_Toc178276507)

[**4.** קטגוריות של NOSQL 14](#_Toc178276508)

[4.1. מבוססי מפתח-ערך (Key-Value Stores): 14](#_Toc178276509)

[4.2. מבוססי עמודות (Wide-Column Stores): 15](#_Toc178276510)

[4.3. מבוססי מסמכים (Document Databases): 17](#_Toc178276511)

[4.4. מבוססי גרפים (Graph Stores): 20](#_Toc178276512)

[**5.** בסיסי נתונים מסוג NoSQL 22](#_Toc178276513)

[5.1. MongoDB: 22](#_Toc178276514)

[5.2. Cassandra: 23](#_Toc178276515)

[5.3. Redis: 24](#_Toc178276516)

[5.4. Neo4J: 25](#_Toc178276517)

[**6.** פתרונות חיפוש וייצוג מידע 27](#_Toc178276518)

[6.1. Elastic Search: 27](#_Toc178276519)

[6.2. RDF: 28](#_Toc178276520)

[**7.** השוואה בין NoSQL ל – SQL 29](#_Toc178276521)

[7.1. יתרונות וחסרונות SQL: 29](#_Toc178276522)

[7.2. יתרונות וחסרונות NoSQL: 31](#_Toc178276523)

[7.3. עקרונות ACID מול BASE: 33](#_Toc178276524)

[7.4. שיקולים בבחירת NoSQL על פני SQL : 34](#_Toc178276525)

[7.5. תיאוריית CAP: 36](#_Toc178276526)

[**8.** החלפת נתונים עבר מסדי נתונים 38](#_Toc178276527)

[8.1. XML: 38](#_Toc178276528)

[8.2. JSON: 40](#_Toc178276529)

[**9.** כלים לביצוע פעולות וחישובים על הנתונים 41](#_Toc178276530)

[9.1. Java Stream: 41](#_Toc178276531)

[9.2. Apache Spark: 43](#_Toc178276532)

[**10.** סיכום ומסקנות 46](#_Toc178276533)

[**11.** ביבליוגרפיה 50](#_Toc178276534)

# תקציר מנהלים

בעולם הטכנולוגיה המודרנית, מסדי נתונים משמשים כבסיס עליו נשענות אינספור אפליקציות ומערכות. מדובר באוספים מאורגנים של נתונים המאפשרים אחסון, שליפה ומניפולציה של מידע בצורות שונות. ממאגרי כתובות פשוטים ועד למערכות ארגוניות מורכבות, מסדי נתונים תומכים כמעט בכל היבט של האינטראקציה הדיגיטלית. האבולוציה של מסדי הנתונים לוותה באבני דרך משמעותיות, כל אחת מהן משקפת את הדרישות הגוברות לניהול ואחסון נתונים. בתחילה, מסדי הנתונים הוגבלו למודלים יחסיים מסורתיים, הידועים בשם SQL (שפת שאילתות מובנית). מערכות אלו, הנשלטות על ידי סכימה קשיחה, מצטיינות בניהול נתונים מובנים עם סכימות ויחסים מוגדרים מראש. עם זאת, הופעת האינטרנט והתפשטות סוגי נתונים מגוונים דרשו גישה גמישה יותר לניהול נתונים. דרישה זו הולידה את מסדי הנתונים מסוג NoSQL (לא רק SQL), שנועדו לטפל בנתונים בלתי מובנים או חצי מובנים עם יכולת גידול וזריזות גבוהה יותר.

במסך זה נציג ניתוח מעמיק של מסדי נתונים מסוג NoSQL. נבחן את הרקע ההיסטורי של מסדי נתונים אלו לעומת מסדי נתונים אחרים, עם דגש על המסד הרלציוני המבוסס SQL. נרחיב על מסד הנתונים הרלציוני מבוססי SQL אשר היווה סטנדרט למסדי נתונים במשך עשרות שנים עם שליטה מוחלטת בשוק מסדי הנתונים, עד להתפתחות תחום ה-NoSQL. בהמשך, נבצע סיווג של מסדי הנתונים מסוג NoSQL ונבחן את קטגוריותיהם השונות, כולל מסדי נתונים מסוג מפתחות-ערך, מסדי מסמכים, מסדי גרפים ומסדי נתונים רחבי-עמודות. כמו כן, נסקור את המסדים הפופולריים בתחום זה, כדוגמת Redis, Neo4j, MongoDB ו-Cassandra. נסקור על פתרונות חיפוש וייצוג מידע ע"י Elastic Search וRDF . בנוסף, נעמוד על ההבדלים המשמעותיים בין NoSQL ל-SQL, תוך התמקדות בחוזקות, חולשות ותחומי יישום מתאימים לכל אחד מהסוגים. ננתח את הגישות השונות לניהול מסדי נתונים רלציוניים בהשוואה לאלו של NoSQL. בסיום, נדון באפשרויות לשילוב והחלפת נתונים בין מסדי נתונים שונים כמו XML ו JSON, ובכלים הזמינים לביצוע פעולות וחישובים על נתונים בהיקפים גדולים כמו Java Stream ו Apache Spark. לבסוף, נציג את מסקנות הניתוח ואת ההמלצות הנובעות ממנו.

העבודה מתארת באופן מירבי את ההשוואה של הגישה של NoSQL למול SQL. כמו כן, מסמך זה מתאר שימושים בקטגוריות שונות של NoSQL תוך שימוש בדוגמאות פופולריות . על ידי הבנת ההבדלים הבסיסיים בין שני הפרדיגמות ובין הקטגוריות השונות, עסקים ומפתחים יכולים לקבל החלטות מושכלות בנוגע לבחירת מסד הנתונים, ולהבטיח ביצועים ויכולת גידול מיטביים עבור האפליקציות שלהם.

# מבוא

## כללי**:**

מסדי נתונים מסוג NoSQL הפכו לנדבך חשוב ביישומים מודרניים, בהם נדרשת גמישות גבוהה לטיפול במידע מובנה, חצי-מובנה ולא-מובנה. מסדי נתונים אלה מותאמים במיוחד לעולמות ה-Big Data, IoT, ומערכות ניהול תוכן (CMS), וזאת בשל יכולתם להתמודד עם כמויות נתונים גדולות ומגוונות תוך שמירה על ביצועים גבוהים וגמישות בעיצוב הסכמה של הנתונים. מידע מובנה מתייחס למידע המאורגן בפורמט קבוע ומוגדר, כמו טבלאות מסדי נתונים עם עמודות ושורות. מסדי נתונים יחסיים (RDB) מתאימים במיוחד לסוג מידע זה בשל היכולת שלהם לנהל סכמה ברורה ויחסים בין נתונים. מידע מובנה חשוב במערכות שדורשות חיפושים מדויקים ושאילתות מורכבות, כמו מערכות פיננסיות או ERP. (Silberschatz et al. 2010).

עם זאת, מסדי נתונים יחסיים עשויים להיות מוגבלים במהירותם, במיוחד כאשר יש צורך בביצוע פעולות הצטרפות (JOIN) מורכבות, דבר שעלול לפגוע בביצועים. מידע חצי-מובנה כולל מידע שבו יש מבנה חלקי, כמו JSON או XML, המאפשר גמישות רבה יותר מאשר בטבלאות מסורתיות. סוג זה של מידע נפוץ במיוחד ביישומים מבוססי רשתות חברתיות או אתרי מסחר אלקטרוני (Silberschatz et al. 2010). מסדי נתונים NoSQL, כדוגמת MongoDB, מותאמים במיוחד לטיפול במידע חצי-מובנה בשל היכולת שלהם להתמודד עם שדות דינמיים וללא צורך בהגדרת סכמה קבועה מראש.

מידע לא-מובנה כולל מידע שאינו בעל פורמט קבוע, וכולל מסמכים, תמונות, סרטונים וקבצי אודיו. מסדי נתונים NoSQL מצטיינים בטיפול במידע לא-מובנה, המהווה את עיקר המידע בעולם הדיגיטלי. לדוגמה, מערכות כמו Couchbase או Hadoop HDFS מאפשרות אחסון ואחזור של כמויות גדולות של מידע לא מובנה במהירות. השילוב של כל שלושת סוגי המידע - מובנה, חצי-מובנה ולא-מובנה - מאפשר לארגונים להתמודד עם האתגרים המגוונים של ניהול מידע מודרני. למשל, יישומי IoT מייצרים נתוני חיישנים לא-מובנים בשילוב עם נתונים מובנים ממסדי נתונים מסורתיים, ומסדי נתונים NoSQL מספקים את הגמישות הנדרשת להתמודדות עם סוגי מידע שונים אלו Tiwari, S. 2011)).

יתרון נוסף של מסדי נתונים NoSQL הוא היכולת לפעול ללא סכמה קבועה מראש, הידועה כ-Schemeless. מסדי נתונים Schemeless מאפשרים גמישות מרבית בעיצוב הנתונים, מכיוון שהם אינם דורשים הגדרת סכמה מראש או שינוי הסכמה כאשר משתנים הנתונים. זה מאפשר למפתחים להוסיף שדות חדשים או לשנות מבני נתונים בקלות מבלי להשפיע על נתונים קיימים או לשנות את הסכמה הכוללת של בסיס הנתונים. היתרון הזה משמעותי במיוחד בסביבות דינמיות כמו מערכות ניהול תוכן או יישומי IoT, בהם דרישות הנתונים עשויות להשתנות בתדירות גבוהה Tiwari, S. 2011)).

היתרון הגדול של מסדי נתונים NoSQL הוא היכולת שלהם לטפל בנתונים בפורמטים מגוונים, ולעדכן או לשנות את הסכמה בקלות, מבלי לפגוע בנתונים קיימים. NoSQL מותאם במיוחד ליישומים בזמן אמת בזכות היכולת לספק ביצועים גבוהים ולבצע קריאות וכתיבות מהירות.

כמו כן, מסדי נתונים רבים מסוג NoSQL הם קוד פתוח, מה שהופך אותם לפתרון חסכוני עבור ניהול נתונים בקנה מידה גדול. עם זאת, מסדי נתונים NoSQL מציבים אתגרים, במיוחד כשמדובר ברגולציות מחמירות כמו ה-GDPR. ארגונים נדרשים לאזן בין הצרכים העסקיים שלהם לבין שמירה על פרטיות הנתונים האישיים, דבר שמוסיף מורכבות לניהול הנתונים (Chen, 2019). בעת בחירת מסד נתונים NoSQL, יש לקחת בחשבון סוגי נתונים, דרישות קנה מידה וצורכי ביצועים, כאשר כל סוג של מסד נתונים NoSQL מתאים למקרי שימוש שונים.

בנוסף, מסדי נתונים מסוג NoSQL ניתנים לשילוב עם שפות תכנות ומסגרות שונות באמצעות ממשקי API וספריות, מה שמאפשר אינטגרציה קלה עם יישומים. בסופו של דבר, מסדי נתונים NoSQL מהווים פתרון יעיל וגמיש לניהול נתונים מודרניים, המאפשרים להתמודד עם כמויות גדולות של מידע דינמי ומגוון, תוך שמירה על ביצועים גבוהים וגמישות בעיצוב המידע. (Chen, 2019).

## נתון מידע וידע:

נתון (Data) הוא ייצוג של עובדה, מספר, רעיון או תמונה. המילה "נתון" נגזרת מהמילה הלטינית "datum," שמשמעותה לתת. (Berg, Seymour, & Goel, 2013) נתון עצמו הוא פריט בסיסי של מידע שאינו מפורש בפני עצמו; זהו רק אוסף של מספרים, מילים או תמונות ללא הקשר או משמעות. לדוגמה, מספרים כמו "25", "שמש" או "חם" הם נתונים. הם אינם מספקים מידע על מהותם או הקשרם בפני עצמם (Boisot & Canals, 2004). הנתונים יכולים להיות אקראיים ולא מסודרים, ולכן הם דורשים עיבוד כדי להפוך למידע שימושי.

מידע (Information) הוא נתון שעבר עיבוד או הקשר, מה שמקנה לו משמעות. כאשר נתונים מאורגנים ומוצגים בצורה שמאפשרת הבנה, הם הופכים למידע. לדוגמה, כאשר אנו מחשבים ומצהירים "הטמפרטורה היום היא 25 מעלות צלזיוס," יש לנו מידע שמקנה הקשר ברור לנתון. מידע מספק משמעות ויכול לשמש כבסיס להחלטות, אך ההבנה המלאה שלו תלויה בידע נוסף של המקבל. לדוגמה, קבלת הודעה מוצפנת עשויה לספק מידע, אך ההבנה המלאה שלה תלויה בידע הקודם של המקבל (Boisot & Canals, 2004).

ידע (Knowledge) הוא המידע המובן והמעובד שנצבר על ידי אדם או מערכת. ידע כולל הבנה, ניסיון ותובנות שנבנו על בסיס מידע. הוא מושפע מהקשרים תרבותיים, חברתיים ואישיים, ומאפשר לאנשים לפרש מידע ולהגיב אליו בצורה אדפטיבית. לדוגמה, אם מישהו יודע שהטמפרטורה של 25 מעלות צלזיוס היא נוחה לפעילות חיצונית, הוא יכול להשתמש במידע הזה כדי לתכנן יום טיול (Boisot & Canals, 2004). ידע נבנה על בסיס המידע, אך הוא גם כולל תובנות שנצברות מהניסיון ומההקשרים השונים שבהם המידע מופיע.

בהקשר של ארגון נתונים, המונח "metadata" מתאר את הנתונים על הנתונים עצמם. נתונים אינם אומרים הרבה כשלעצמם, אך כאשר הם שייכים לקבוצה ומאורגנים עם metadata, הם מקבלים יותר משמעות. זהו מידע נוסף שמסביר את הקשר והמאפיינים של הנתונים. המונח "מסד נתונים" מתאר את האיסוף והארגון של נתונים בצורה שמאפשרת גישה נוחה ושימושית, והוא הכרחי כדי לארגן, לאחסן ולהפוך את הנתונים לזמינים (Berg, Seymour, & Goel, 2013)

## מסד נתונים:

מסד נתונים הוא אוסף עצמאי של רשומות משולבות. רשומה היא ייצוג של אובייקט פיזי או קונספטואלי כלשהו. מסד נתונים מתאר את עצמו בכך שבוא מכיל תיאור של המבנה שלו. תיאור זה נקרא metadata - data about the data. מסד הנתונים משולב בכך שהוא כולל את היחסים בין פריטי נתונים, וכן כולל את פריטי הנתונים עצמם. (Kroenke & Auer, 2007).

## רקע והיסטוריה של מסדי נתונים:

בני האדם התחילו לאחסן מידע לפני זמן רב. לכן, במהלך השנים, אנשים הבינו שהם צריכים את האמצעים לאחסון נתונים ולשמור על קבצי הנתונים לשם אחזורים מאוחר יותר. לכן, הם ניסו למצוא דרכים לאחסן, לאנדקס, ולאחזר נתונים. עם הופעת המחשבים, עולם מסדי הנתונים השתנה במהירות, מה שהפך את האיסוף ותחזוקת מסד הנתונים למשימה קלה, חסכונית ויעילה יותר .(Date, 2003)

בתקופות העתיקות, מערכות מסדי נתונים מורכבות פותחו על ידי משרדי ממשלה, ספריות, בתי חולים וארגונים עסקיים, וחלק מהעקרונות והקונספטים הבסיסיים ששימשו בבניית מערכות אלו עדיין בשימוש כיום. בשנות ה-60, עם העלייה בפופולריות של מחשבים והפיכתם לזולים יותר לשימוש, התחולל שינוי משמעותי בתחום מסדי הנתונים (DuCharme, 2012).

המונח "מסד נתונים" החל להיות נפוץ בעקבות הזמינות של אחסון בגישה ישירה כמו דיסקים ותופים, מה שאפשר שימוש אינטראקטיבי משותף בניגוד לעיבוד באצוות (Batch Processing) שהתבסס על סרטים מגנטיים. בתקופה זו פותחו שני מודלים עיקריים של מאגרי נתונים: מודל הרשת(Conference on Data System Language ) CODASYL והמודל ההיררכי ((Information Management System )IMS. הדור הראשון של מערכות מאגרי נתונים היה ניווטי, בו יישומים גשו לנתונים באמצעות מצביעים בין רשומות. ההתמקדות הייתה בעיבוד הרשומות ולא במבנה הכללי של המערכת, מה שדרש מהמשתמשים להכיר את המבנה הפיזי של המאגר. אחד ממאגרי הנתונים המוצלחים ביותר באותה תקופה היה מערכת SABRE, ששימשה את IBM לניהול הזמנות עבור American Airlines (Bercich, 2002).

המודל הרשת וההיררכי, לא היו מסוגלים להיענות לדרישות המתפתחות של זמנו, מה שהוביל לצורך במודלים מתקדמים יותר (Berg, Seymour, 2013; Goel). בשנות ה-70, אדגר קוד E.F. Codd פיתח את מודל מסד הנתונים הרלציוני, אשר שינה את הגישה המסורתית בכך שהדגיש חיפוש נתונים לפי תוכן במקום באמצעות מעקב אחר קישורים. ניתן להגדיר את המערכת שלו באמצעות שני מונחים (Taylor, 2000). "מופע" (Instance) הוא טבלה עם שורות ועמודות, ו"סכמה" (Schema) מציינת את המבנה, כולל שם הקשר, השם והסוג של כל עמודה. במהלך התקופה הזו, נטבע המונח "מערכת ניהול מסדי נתונים רלציוניים" (RDBMS - Relational Database Management System).

בשנות ה-80 החלה המסחור של מערכות הרלציוניות, כאשר הגידול ברכישת מחשבים הזניק את שוק מאגרי הנתונים עבור עסקים. שפת ה-SQL (Structured Query Language) הפכה לסטנדרט בינלאומי. . (Taylor, 2007) DB2 הפכה למוצר המרכזי של IBM. המודלים הרשתיים וההיררכיים איבדו את קסמם ולא פותחו מערכות חדשות על פי מודלים אלו, אך חלק מהמערכות הישנות עדיין בשימוש כיום. פיתוח המחשב האישי של IBM הביא לעלייתן של חברות ומוצרים רבים בתחום מאגרי הנתונים, כמו RIM, RBASE 5000, PARADOX, OS/2 Database Manager, Dbase III, IV (שמאוחר יותר הפכו ל-FoxBASE ול-Visual FoxPRO) ו-Watcom SQL. כל המערכות הללו הוצגו בשנות ה-80 והיו מבוססות על המודל הרלציוני.

באותה תקופה פותח הקונספט של מאגרי נתונים מונחי אובייקטים. המונח "מערכת ניהול מאגרי נתונים מונחי אובייקטים" הופיע לראשונה סביב שנת 1985. מאגר נתונים מונחה אובייקטים, המכונה גם OODBMS (Object-Oriented Database Management System), הוא מאגר בו המידע מיוצג בצורה של אובייקטים, כמו בשפות תכנות מונחות אובייקטים (Haadi,2010).עם זאת, מעבר למערכות OODBMS היה מאתגר בשל חוסר גמישות וקשירה לשפות תכנות ספציפיות. כדי להתגבר על הבעיות הללו, פותח מודל המאגר האובייקטי-רלציוני בתחילת שנות ה-90.

התפתחות מרכזית נוספת שהתרחשה בסוף שנות ה 90 הייתה הצגת שפת התגיות המורחבת XML . (DuCharme, 2012) XML היא שפת תגיות שמגדירה קבוצה של כללים לקידוד מסמכים בפורמט שהוא קריא גם לבני אדם וגם למכונות. עם התקרבות המאה ה-21, בשנת 2000 הייתה השפעה מתופעת Y2K ("באג 2000") על חידושים טכנולוגיים, שהובילה ארגונים לשדרג את המערכות שלהם. חברות גדולות כמו IBM, Microsoft ו-Oracle הפכו לשחקניות דומיננטיות בשוק מסדי הנתונים, והניעו חידושים ששיפרו את שלמות הנתונים, אבטחת המידע ויעילות המערכת הכללית.

התפתחות מרכזית נוספת חשובה שהתרחשה הייתה NoSQL . המונח NoSQL הוצג לראשונה על ידי קרלו סטרוזי (Carlo Strozzi) ב-1998 כדי לתאר מסד נתונים רלציוני קל משקל ופתוח שלא חשף ממשק SQL סטנדרטי. מאוחר יותר, המונח הושאל על ידי אריק אוונס (Eric Evans) כדי לתאר את הצמיחה של מאגרי נתונים לא-רלציוניים ומבוזרים שנמנעו מהתחייבויות ACID (אטומיות, עקביות, בידוד, עמידות) שמאפיינות את מערכות מסדי הנתונים הרלציוניות הקלאסיות ובמקומו יופעל עקרון ה BASE (זמינות בסיסית, מצב רך ועקביות). NoSQL נוצר בעקבות התקדמות האינטרנט והמחשוב בענן שיצרו צורך במסדי נתונים המיועדים לאחסון ועיבוד כמויות גדולות של נתונים בצורה יעילה, תוך שמירה על ביצועים גבוהים בקריאה ובכתיבה. מסדי הנתונים המסורתיים, מבוססי SQL, נתקלו באתגרים חדשים, בעיקר ביישומים רחבי היקף עם תעבורה גבוהה, כגון מנועי חיפוש ורשתות חברתיות (Han, Haihong, Le, & Du, 2011).

במצבים אלו, מסדי הנתונים היחסיים לא הספיקו, והצורך הוביל לפיתוח מסדי נתונים מסוג NoSQL(Not Only SQL). מסדי נתונים אלו נותנים אחסון וגישה למידע שאינו ממודל טבלאי יחסי . בסיסי נתונים מסוג NoSQL הופכים לנפוצים במערכות Big Data וכן המערכות בזמן אמת והם מציעים סקלביליות וגמישות על ידי תמיכה בנתונים בלתי מובנים וללא צורך בסכמה קבועה. פתרונות כמו MongoDB ו-Cassandra אפשרו טיפול בנפחי נתונים גדולים ומגוונים, והתאימו במיוחד ליישומים מודרניים כגון פלטפורמות מדיה חברתית ומסחר אלקטרוני. (Berg, Seymour, 2013; Goel).

# מסד נתונים רלציוני

## תיאור מסד הנתונים:

מסד נתונים רלציוני מאפשר ארגון, אחסון וגישה יעילה למידע רב ומגוון. בסיס נתונים רלציוני הוא אוסף של נתונים מאורגנים בטבלאות, שורות ועמודות. כל טבלה מייצגת ישות מסוימת (כגון לקוחות, מוצרים, הזמנות), והקשרים ביניהן מוגדרים באמצעות מפתחות. המבנה של מסד נתונים זה מאפשר אחסון, ניתוח וגישה יעילה למידע רב ומגוון. מסדי נתונים רלציוניים, המהווים את עיקר מאגרי המידע הנמצאים בשימוש כיום (Harrington, 2016).

המודל הרלציוני לבסיסי נתונים פותח לראשונה על ידי אדגר קוד בשנת 1970. מאז, המודל התפתח והפך לסטנדרט בתעשיית המחשוב. בבסיס נתונים רלציוני כל טבלה מכילה מידע על ישות מסוימת, כאשר כל שורה מייצגת רשומה אחת בטבלה. כל עמוד מייצג תכונה של הישות. המבנה הטבלאי מאפשר ארגון נתונים בצורה לוגית וברורה. ניתן להוסיף, לשנות ולמחוק נתונים בקלות.

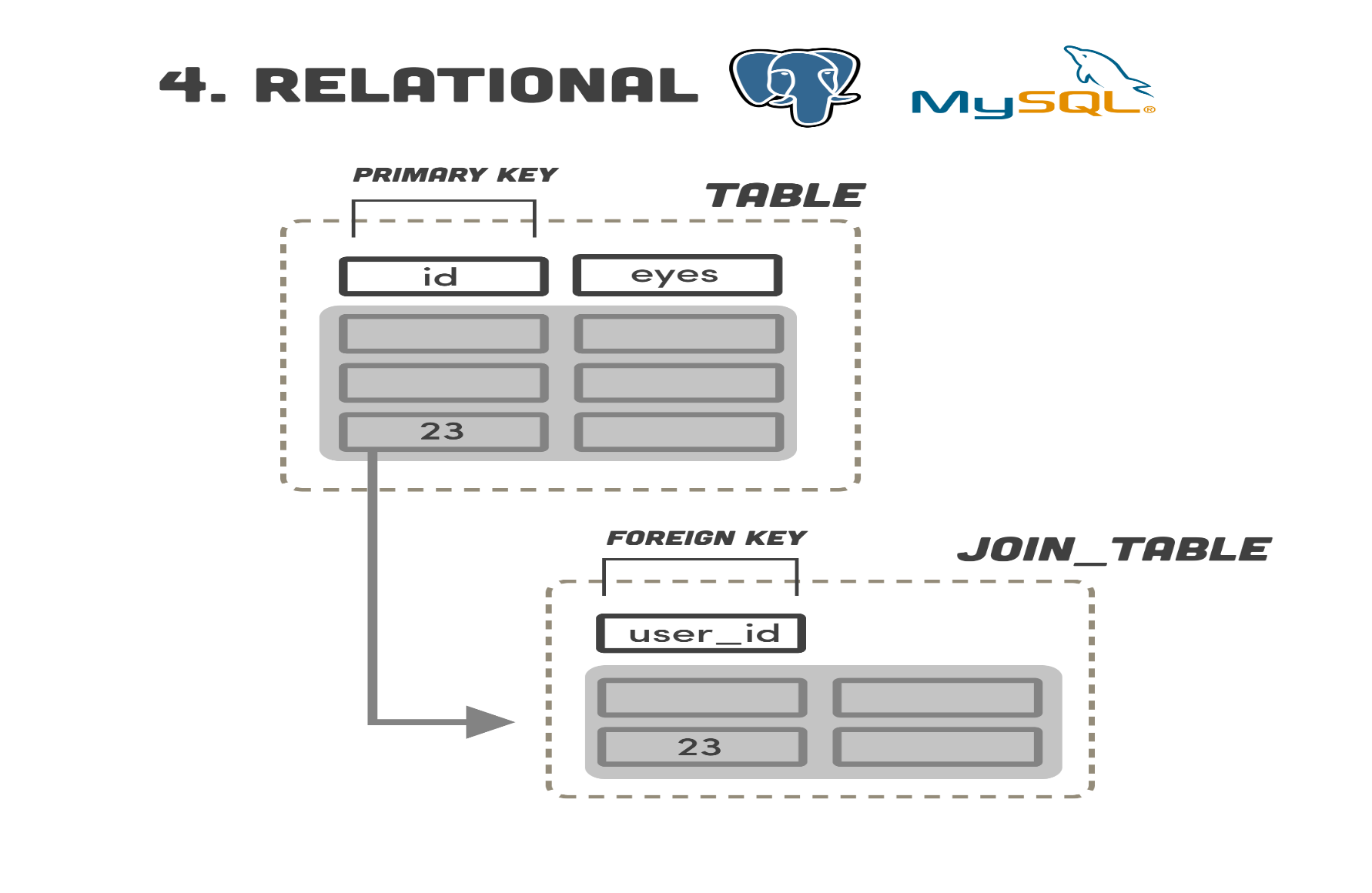
מערכות ניהול מסדי נתונים רלציוניים RDBMS- Relational Database Management System מותאמות לבצע שאילתות מורכבות במהירות. שלמות הנתונים בבסיס הנתונים מבטיחה דיוקם ועקביותם. יש לציין ששפת SQL -Structured Query Language משמשת לתפעול מסדי נתונים רלציוניים. בסיסי נתונים רלציוניים מסתמכים על מנגנונים שונים כדי לשמור על שלמות הנתונים, לאפשר גישה מהירה וליצור קשרים בין טבלאות. שלושת המרכיבים המרכזיים שתומכים בשלמות הנתונים הם: מפתח ראשי, מפתח זר ואינדקס. Silberschatz et al 2010))

מפתח ראשי (Primary Key) מוגדר כלפחות עמודה אחת או יותר בטבלה שמזהות באופן ייחודי כל שורה בטבלה. כלומר, ערך המפתח הראשי אינו חוזר על עצמו בשום שורה אחרת. השימוש במפתחות ראשיים נועד לזיהוי ייחודי לכל רשומה, משמש כבסיס ליצירת קשרים בין טבלאות באמצעות מפתחות זרים ומבטיח שאין שתי שורות זהות בטבלה. את המפתח הראשי מגדירים בעת יצירת הטבלה או לאחר מכן באמצעות שפת SQL. מפתח ראשי נועד למטרות שלמות נתונים בטבלאות, מונע כפילויות וגם משמש כבסיס לאינדקסים, מה שמאיץ שאילתות. החיסרון של מפתח ראשי הוא שלאחר הגדרה, שינוי המפתח הראשי יכול להיות תהליך מסובך. חובה להגדיר מפתח ראשי בכל טבלה כדי לשמור על שלמות הנתונים ולשפר את ביצועי השאילתות Garcia-Molina et al. 2008)).

מפתח זר (Foreign Key) הינו עמודה בטבלה אחת המצביעה על מפתח ראשי בטבלה ועמודה שנייה היוצרת קשר בין שתי טבלאות. מגדירים את המפתח הזר בעת יצירת הטבלה או לאחר מכן באמצעות שפת SQL, תוך התייחסות למפתח הראשי בטבלה המקושרת. שימוש במפתח זר בבסיסי נתונים רלציוניים נועד לקשר בין ישויות שונות במסד הנתונים וגם מבטיח שהערכים במפתח הזר קיימים במפתח הראשי בטבלה המקושרת. מפתח זר מאפשר ייצוג של קשרים בין ישויות שונות ומונע מצבים לא עקביים. עם זאת חשוב לקחת בחשבון ששימוש במפתח זר ביצירת קשרים מורכבים בין טבלאות יכול להקשות על התכנון והתחזוקה של מסד הנתונים. Garcia-Molina et al. 2008))

אינדקס (Index) הינו מבנה נתונים נוסף שמאפשר גישה מהירה לנתונים בטבלה על בסיס ערך או קבוצת ערכים בעמודה מסוימת. מגדירים אינדקס על עמודה אחת או יותר בטבלה. הגדרת אינדקס מאפשרת חיפוש מהיר של נתונים על פי קריטריונים מסוימים ומאפשרת סידור נתונים לפי סדר מסוים. אינדקס משפר משמעותית את מהירות השאילתות, במיוחד עבור טבלאות גדולות. עם זאת שימוש באינדקסים תופס מקום לא מבוטל נוסף בדיסק, כאשר עדכון אינדקסים לאחר שינויים בנתונים יכול להאט את ביצועי הכתיבה. כמו כן, הגדרת האינדקס דורשת תכנון מדוקדק כדי למנוע יצירת אינדקסים מיותרים. יש ליצור אינדקסים על עמודות שעליהן מבוצעות שאילתות חיפוש תכופות או על עמודות המשמשות כקריטריון מיון. Silberschatz et al 2010))

לדוגמה Users Table זוהי הטבלה המרכזית המכילה פרטי משתמשים. בכל שורה יש מידע על משתמש אחד, והמפתח הראשי (Primary Key) הוא העמודה id, שהוא מזהה ייחודי לכל משתמש. Orders Table: טבלה זו מכילה פרטים על הזמנות שבוצעו על ידי המשתמשים. היא מקושרת לטבלת ה-Users באמצעות עמודת user\_id, שהיא המפתח הזר (Foreign Key) ומפנה ל-id בטבלת ה-Users.



(Fireship. (n.d.). Top seven database paradigms. Fireship.io)

## שפת SQL:

SQL - Structured Query Language היא שפת מחשב שתוכננה במיוחד לטיפול ועיבוד מידע במסדי נתונים יחסיים. היא מאפשרת לבצע מגוון רחב של פעולות על נתונים המאוחסנים בטבלאות, כמו חיפוש והצגת מידע ספציפי מתוך הטבלאות, שינוי, הוספה ומחיקה של נתונים קיימים, הגדרת מבנה חדש לאחסון נתונים ושינוי מבנה הטבלאות, יצירת אינדקסים ועוד. שפת SQL פותחה בחברת IBM בתחילת שנות ה-70. השפה הוצגה לראשונה באופן רשמי בשנת 1974. הגרסה הראשונה של SQL, המכונה Structured English Query Language, נוצרה ככלי פנימי בחברת IBM. המטרה הייתה לפתח שפה פשוטה ואינטואיטיבית לאינטראקציה עם מסדי נתונים. Silberschatz et al. 2010).)

בשנת 1986 הארגון ANSI - American National Standards Institute קבע את SQL כתקן רשמי, עם מה שנודע כ-ANSI SQL-86. תקן זה היה נקודת ציון חשובה בהבניית השפה. בשנת 1999 יצא גרסה של שפה-SQL3 שכללה תוספות כמו תמיכה באובייקטים, טריגרים ויכולות מתקדמות אחרות. התקנים של השפה שיצאו לשוק בין השנים 2003 ל 2006 הכילו שיפורים נוספים, כולל הרחבות לפונקציות ומבני נתונים חדשים. במהלך השנים, SQL עברה התפתחויות רבות והפכה לתקן תעשייתי המיושם על ידי רוב מערכות ניהול מסדי הנתונים היחסיים – RDBMS. ארגונים כמו ANSI ו-ISO מגדירים סטנדרטים ל-SQL, ומבטיחים תאימות בין מערכות שונות.

שפת SQL הינה הסטנדרטית לניהול מסדי נתונים יחסיים, והיא נתמכת על ידי רוב מערכות ניהול מסדי הנתונים הקיימות כמו MySQL, PostgreSQL, Oracle ועוד. מדובר בשפת תכנות גמישה המאפשרת לבצע מגוון רחב של פעולות, החל משאילתות פשוטות ועד למניפולציות מורכבות על הנתונים. SQL משמשת לנהל ולתחזק נתונים בתוך מסדי נתונים. היא מאפשרת למשתמשים להוסיף, לעדכן, למחוק ולשלוף נתונים מתוך הטבלאות במערכת (Malik, 2019).

SQL מאפשרת לבצע שאילתות מורכבות על הנתונים. באמצעות פקודות כמו select, ניתן לשלוף נתונים מסוימים מתוך מסד הנתונים ולהפיק דוחות וסטטיסטיקות. SQL לא רק מטפלת בנתונים אלא גם במבנה של מסד הנתונים עצמו. פקודות כמו create table ו- alter table מאפשרות יצירה ושינוי של טבלאות ומבנים אחרים במערכת. SQL מספקת כלים לניהול גישת משתמשים ופרופילים, מה שמאפשר שליטה על מי יכול לקרוא, לכתוב או לשנות נתונים.

SQL כוללת כלים לייעול ביצועים של שאילתות, כולל אינדקסים ואסטרטגיות אופטימיזציה שמסייעות לשפר את מהירות החיפושים ושליפת הנתונים. בשל הפשטות והיכולת של SQL לנהל נתונים בצורה מאוד יעילה ומדויקת, היא נשארת כלי עיקרי בכל תחום הדורש ניהול נתונים. SQL היא אחת השפות הנפוצות ביותר לניהול מסדי נתונים. היא נמצאת בשימוש נרחב בכל תחום בו יש צורך בניהול נתונים בצורה מבוקרת ומסודרת. השפה נמצאת בשימוש אצל מפתחים לפיתוח יישומים ותוכנות המשתמשות במסדי נתונים, אנליסטים לניתוח נתונים ולהפקת דוחות, חוקרי נתונים לניתוח וביצוע מדע נתונים ועוד. Silberschatz et al. 2010).)

## אלגברה רלציונית:

אלגברה רלציונית היא למעשה שפה פורמלית שמשמשת לתיאור וביצוע פעולות על נתונים מאורגנים בטבלאות. היא מהווה את הבסיס התיאורטי של מסדי הנתונים היחסיים, שבהם הנתונים מיוצגים בצורה של טבלאות. במונחים של אלגברה רלציונית בטבלה, כל שורה מייצגת רשומה (או שורה) והיא מכילה ערכי נתונים שונים. כל עמודה מייצגת תכונה (או שדה) משותפת לכל הרשומות. כל טבלה מתוארת כיחס Relation. כל יחס כולל מספר אטומים Tuples שכל אחד מהם הוא שורה בטבלה, וכל שורה כוללת ערכים בעמודות או שדות של הטבלה. שתי גישות הוצעו בספרות להרחבת האלגברה הרלציונית באופן זמני: חתימות זמן של שורות tuple time-stamping וחתימות זמן של תכונות attribute timestamping ( Dey, D, 1996).

אלגברה רלציונית מהווה את הבסיס התיאורטי לניהול מסדי נתונים יחסיים, ומספקת מגוון רחב של פעולות שמאפשרות למשתמשים ולמפתחים לעבוד עם נתונים בצורה גמישה ויעילה. היא מספקת את היסודות לשפות שאילתות כמו SQL, ותורמת להבנה מעמיקה של הדרך בה נתונים מנוהלים ומעובדים.

האלגברה הרלציונית כוללת מספר פעולות עיקריות, שכל אחת מהן פועלת על יחסים ומחזירה יחסים חדשים. הפעולה Projection בוחרת עמודות מסוימות מתוך הטבלה, ומחזירה טבלה חדשה הכוללת רק את העמודות שנבחרו. הפעולה Selection בוחרת שורות מסוימות מתוך הטבלה על פי קריטריונים מסוימים. התוצאה היא טבלה הכוללת את השורות שעומדות בתנאי הסינון. הפעולה Join מחברת בין שתי טבלאות על פי תנאי התאמה בין העמודות שלהן. החיבור יוצר טבלה חדשה הכוללת את כל השורות שמבצעות התאמה בין הטבלאות. ישנם סוגים שונים של חיבורים, כמו Inner Join ו- Outer Join המבדילים בין סוגי ההתאמות הנעשות. פעולה Union מאחדת שתי טבלאות עם אותה מבנה ומחזירה טבלה חדשה הכוללת את כל השורות מהטבלאות המקוריות, תוך הימנעות משורות כפולות. ( Dey, D, 1996).

פעולה Difference מוצאת את השורות שמופיעות בטבלה אחת ולא מופיעות בטבלה השנייה. התוצאה היא טבלה חדשה עם שורות מהטבלה הראשונה שמופיעות רק שם. פעולה Intersection מוצאת את השורות המופיעות בשתי הטבלאות. התוצאה היא טבלה חדשה. פעולה Renaming משמשת לשנות את שמות העמודות או הטבלאות, או לשנות את השם של יחס מסוים לשם חדש. היא מאפשרת לתת שמות שונים ליחסים ולשדות לצורך הבהרה או פשוטות בעבודה עם נתונים.

באלגברה הרלציונית כל פעולה מחזירה יחס חדש, מה שמאפשר לבצע שרשרות של פעולות בצורה מתודית. הפעולות נתפסות כטכניות ומדויקות, עם תכונות מתמטיות ברורות, מה שמאפשר ניתוחים מעמיקים של נתונים. האלגברה הרלציונית מספקת את הבסיס התאורטי לשפות שאילתות, ומאפשרת פיתוח ותכנון של מערכות מסדי נתונים עם עקרונות ברורים ומוגדרים. Silberschatz et al. 2010))

## נורמליזציה:

נורמליזציה היא תהליך ארגון נתונים במסד נתונים בצורה יעילה ומבוזרת, כך שיימנעו סתירות, חוסר עקביות והשפעות שליליות על ביצועי המערכת. נורמליזציה מונעת היווצרות נתונים כפולים כאשר כל פריט מידע מופיע רק במקום אחד, מה שמפחית את הסיכוי לטעויות בעת עדכון הנתונים. היא משפרת שלמות הנתונים כאשר נתונים קשורים נשמרים יחד, מה שמבטיח עקביות ושלמות. נורמליזציה מביאה לגמישות בשינויים במבנה הנתונים ויכולה להתבצע בקלות יחסית, ללא השפעה משמעותית על שאר המערכת. בנוסף היא גורמת לשיפור ביצועים בגישה למבנה נתונים ומאפשרת לבצע שאילתות בצורה יעילה יותר. Silberschatz et al. 2010)).

תיאוריית נורמליזציה של מסדי נתונים מציעה שיטות כיצד להפחית נתונים מיותרים וכך לתת מענה לבעיות נפחי מידע במסדי נתונים. חוקרים רבים מפתחים שיטות תלות לא רצויה בנתונים מיותרים. כדי לטפל בבעיות אלה, החוקרים הציעו את תיאוריית המערכות המנורמלות שהיישום המלא שלהן משחרר מערכות מהשפעות קומבינטוריות וכך למעשה שינוי המבנה של בסיס הנתונים אינה תלויה בגודל המערכת. (Eessaar, E. 2016). נורמליזציה היא תהליך חשוב, אך יש לשקול את היתרונות והחסרונות שלו.

בדרך כלל, מבצעים נורמליזציה מלאה (עד לרמה השלישית או הרביעית) כאשר השלמות והעקביות של הנתונים הן קריטיות. עם זאת, במקרים מסוימים, כמו במערכות OLAP (Online Analytical Processing), נרצה לוותר על חלק מהנורמליזציה כדי לשפר את ביצועי השאילתות. ישנן מספר רמות של נורמליזציה, כאשר כל רמה מוסיפה אילוצים נוספים על המבנה של הטבלאות: נורמליזציה מתבצעת בשלבים, כאשר כל שלב נקרא פורמט נורמלי. לכל פורמט יש דרישות משלו המייעלות את מבנה הנתונים.

להלן פירוט של הפורמטים הנורמליים הנפוצים:

פורמט נורמלי ראשון (1NF) - דרישות: כל טבלה צריכה להיות במצב של פורמט נורמלי ראשון אם כל שדה בטבלה מכיל ערכים אטומיים (בלתי ניתנים לחלוקה) ואין חזרות שורות. מטרת הפורמט: להסיר קבוצות נתונים חוזרות ונתונים מכוננים. כל שדה צריך להכיל ערך יחיד, ולא רשימות או ערכים מרובי ערכים Silberschatz et al. 2010)).

פורמט נורמלי שני (2NF): טבלה תהיה בפורמט נורמלי שני אם היא כבר נמצאת ב-1NF, וכל שדה לא מפתחי (non-key) בטבלה תלוי לחלוטין במפתח הראשי. מטרת הפורמט: להסיר תלות חלקית, כלומר כל שדה לא מפתחי צריך להיות תלוי בכל חלק מהמפתח הראשי, ולא רק בחלק ממנו Silberschatz et al. 2010)).

פורמט נורמלי שלישי (3NF) - דרישות: טבלה תהיה בפורמט נורמלי שלישי אם היא כבר נמצאת ב-2NF, וכל שדה לא מפתחי לא תלוי בשדות לא מפתחיים אחרים (כלומר, אין תלות מעבר). מטרת הפורמט: להסיר תלות. אם שדה A תלוי בשדה B ושדה B תלוי בשדה C, אז שדה C צריך להיות תלוי ישירות במפתח הראשי ולא ב- B. Silberschatz et al. 2010)).

פורמט נורמלי שלישי עם תוספות (BCNF): פורמט נורמלי זה הוא הרחבה של 3NF. טבלה נמצאת ב-BCNF אם היא נמצאת ב-3NF וכל מפתח הוא מפתח ראשי. מטרת הפורמט: להבטיח שכל תלות פונקציונלית היא בין מפתחות ראשיים בלבד, ולא תלות בתתי קבוצות אחרות של מפתחות. Silberschatz et al. 2010)).

פורמט נורמלי של רביעי (4NF): טבלה תהיה בפורמט נורמלי רביעי אם היא כבר נמצאת ב-BCNF, ולא קיימת תלות מרובת ערכים. מטרת הפורמט: להסיר תלות מרובת ערכים, שבה שדה בטבלה כולל יותר מקבוצת ערכים אחת שתלויה במפתח הראשי. Silberschatz et al. 2010)).

פורמט נורמלי של חמישי (5NF): טבלה תהיה בפורמט נורמלי חמישי אם היא כבר נמצאת ב-4NF, וכל תלות התפרקות אפשרית נפתרת. מטרת הפורמט: להסיר תלות הנובעת מהתפרקות של טבלה למספר טבלאות נוספות. התמקדות היא באי-תלות בין ערכים מתפרקים Khodorovskii 2002)).

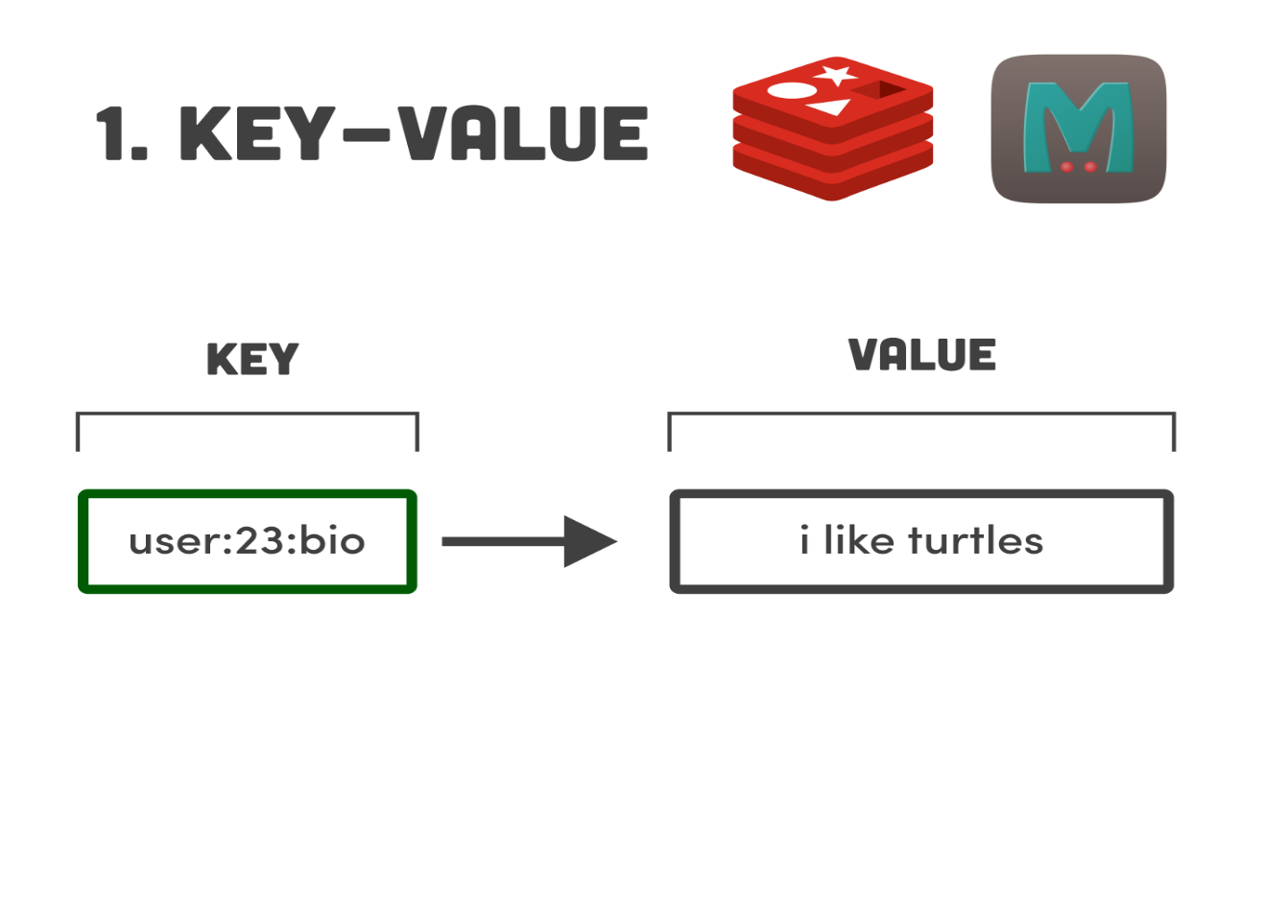
פורמט נורמלי של שלישי (6NF): טבלה תהיה בפורמט נורמלי שלישי אם היא כבר נמצאת ב-5NF, וכל שדה נפרד מאובטח לחלוטין. מטרת הפורמט: להתמקד בתלויות המסוימות בכל מיני מקרים, במיוחד כאשר מדובר על תלויות סבוכות שמגיעות למצב של פירוק נוסף Khodorovskii 2002)).

# קטגוריות של NOSQL

## מבוססי מפתח-ערך (Key-Value Stores):

Key-Value Stores הם סוג של מסדי נתונים לא-יחסיים (NoSQL) המתמקדים באחסון נתונים בצורה של זוגות מפתח-ערך. בכל מערכת כזו, כל ערך (Value) משויך למפתח (Key) ייחודי, המאפשר גישה מהירה ויעילה לנתונים. המפתח משמש כאינדקס שמקשר בין הערך למפתח, ומכיוון שהוא ייחודי, הוא מבטיח גישה מהירה ומדויקת לערכים המאוחסנים. חלק מבסיסי Key-Value Stores מאפשרים להגדיר סוג (TYPE) לערכים כמו: Integer, String .

לדוגמה דיאגרמת נתונים של מפתח (user) וערך (I like turtles)



) Fireship. (n.d.). Top seven database paradigms. Fireship.io)

המאפיין המרכזי של Key-Value Stores הוא הפשטות שלהם. הם מתמקדים באחסון נתונים בצורה של זוגות מפתח-ערך, מבלי להידרש להגדרת סכמות או מבני נתונים מורכבים. השימוש במבנה הפשוט הזה מאפשר ביצועים גבוהים במיוחד, בעיקר עבור פעולות קריאה וכתיבה, מכיוון שאין צורך בעיבוד נוסף של מבני נתונים או ביצוע שאילתות מורכבות. Osemwegie et al., 2018)).

אחד היתרונות המרכזיים של Key-Value Stores הוא ביצועים גבוהים במיוחד בגישה לנתונים. דוגמה לכך היא Redis, מסד נתונים המהווה חלק מהקטגוריה הזאת ומציע ביצועים גבוהים מאוד בזכות אחסון הנתונים בזיכרון (In-Memory), שמפחית את זמני הגישה לנתונים. שימושים אופייניים של Key-Value Stores כוללים מערכות קאשינג (Caching), שבהן ניתן לאחסן ערכים יקרים להשגה לפי מפתחות בודדים, מה שמייעל את מהירות הפעולה של אפליקציות.

בנוסף לביצועים הגבוהים, Key-Value Stores מציעים גמישות רבה בעבודה עם נתונים לא מובנים. המפתחים יכולים לאחסן סוגים שונים של נתונים מבלי להידרש להגדיר מבנה קבוע מראש. זה מקנה להם גמישות רבה ועוזר להם להתאים את מסד הנתונים לצרכים משתנים של האפליקציה.

עם זאת, ישנם חסרונות בשימוש ב-Key-Value Stores. הם אינם תומכים בשאילתות מורכבות או בקשרים בין אובייקטים, מה שעשוי להגביל את השימוש בהם במקרים שבהם יש צורך בניתוח נתונים מתקדם או בקשרים בין טבלאות שונות. בנוסף, המוגבלות של השימוש במפתחות בלבד פוגמת ביכולת לבצע שאילתות מורכבות על הנתונים, מה שעלול להוות בעיה במצבים שבהם נדרשות שאילתות עם תנאים מסובכים. Osemwegie et al., 2018)).

למרות החסרונות, Key-Value Stores נחשבים לפתרון יעיל במיוחד עבור יישומים שמצריכים ביצועים גבוהים ויכולת הרחבה. הם מהווים חלק מרכזי במערכות מידע מודרניות ומשמשים בעיקר במצבים שבהם דרושים זמינות גבוהה ותגובה מהירה, כמו אפליקציות אינטרנטיות גדולות ומערכות שדורשות מהירות וביצועים גבוהים. הם מתאימים במיוחד ליישומים שבהם נתונים הם יחסית פשוטים ולא דורשים ניתוח מתקדם או קשרים בין אובייקטים.

## מבוססי עמודות (Wide-Column Stores):

Wide-Column Stores (WCS) הם חלק מהקטגוריה הרחבה של מסדי נתונים NoSQL, והם מיועדים לטיפול בנתונים גדולים (Big Data) הדורשים גמישות מבנית וביצועים גבוהים. מסדי נתונים אלה פותחו במקור כדי להתמודד עם האתגרים של אחסון נתונים בכמויות עצומות בסביבות שבהן יש צורך לקרוא ולכתוב נתונים במהירות גבוהה תוך שמירה על יכולת להתרחב בצורה אופקית – כלומר להוסיף עוד שרתים ולהפיץ את הנתונים ביניהם כדי לעמוד בעומסים הולכים וגדלים.

ב-Wide-Column Stores, הנתונים מאורגנים במבנה של "משפחות עמודות" (Column Families), שבמובן מסוים דומים לטבלאות במסדי נתונים יחסיים, אך הם גמישים יותר. כל "משפחת עמודות" יכולה להכיל מספר שונה של עמודות עבור כל שורה, וכל עמודה יכולה להחזיק סוגי נתונים שונים. היכולת הזאת מאפשרת התאמה מהירה לשינויים בדרישות היישום או באופי הנתונים, מה שהופך את Wide-Column Stores לבחירה אטרקטיבית עבור יישומים הדורשים גמישות מבנית, כמו מערכות ניהול תוכן, רשתות חברתיות, ויישומים עסקיים המייצרים נתונים משתנים באופן תדיר (Bruckner, 2012).

במסד נתונים מסוג Wide-Column Store, כל מסד נתונים מכיל שורות (Rows), כאשר כל שורה מזוהה על ידי מפתח שורה ייחודי (Row Key). מפתח השורה מפנה אל משפחת עמודות (Column Family) אחת או יותר, שבתוכן יש מספר רב של עמודות (Columns). כל עמודה מכילה שם עמודה (Column Name) ו-ערך עמודה (Column Value), ולעיתים גם חותמת זמן (Timestamp) לצורך ניהול גרסאות של נתונים.

דוגמה למבנה,

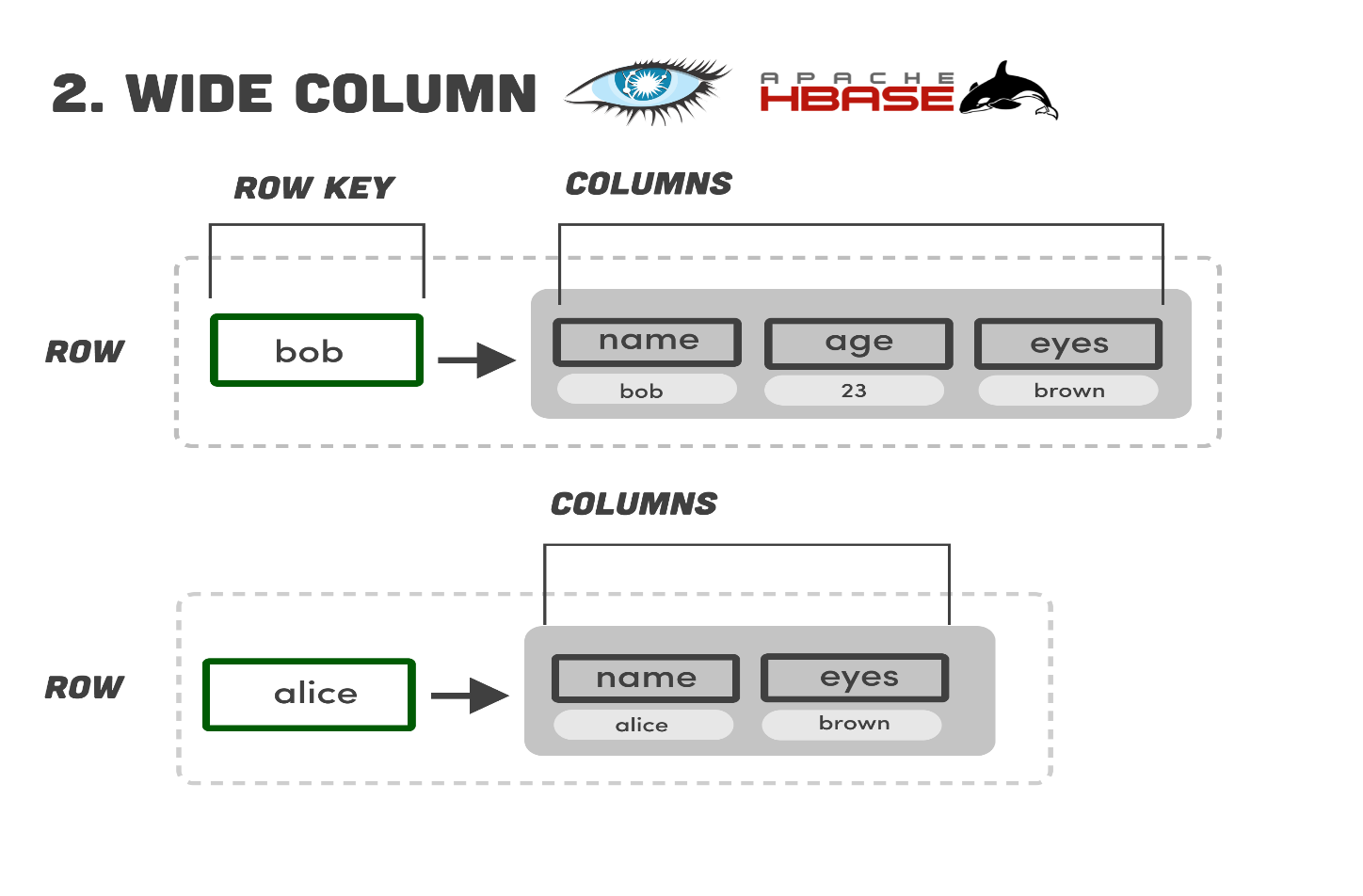
Row Key: bob (מזהה משתמש)

משפחת עמודות: UserDetails

עמודה: name = bob

עמודה: age = 23

עמודה: eyes = brown



Fireship. (n.d.). Top seven database paradigms. Fireship.io))

היתרון המרכזי של WCS הוא הביצועים הגבוהים בכתיבה וקריאה של נתונים, הנובעים מפיזור הנתונים על פני מספר שרתים. כל שרת מאחסן רק חלק קטן מהנתונים, כך שכל פעולת קריאה או כתיבה מתבצעת במהירות רבה יחסית. פיזור זה גם מפחית את הסיכון לכשלי שרת, מכיוון שהנתונים מגובים במספר שרתים, מה שמאפשר זמינות גבוהה. לדוגמה, Cassandra, מערכת WCS פופולרית, נבנתה כדי לפעול בסביבות מבוזרות, ומסוגלת להתמודד עם דרישות של ביצועים גבוהים וזמינות מתמשכת גם כאשר חלק מהשרתים נופלים. הגמישות הזאת מאפשרת למערכת להתמודד עם עומסים כבדים מאוד, והיא מתאימה במיוחד לעבודה עם נתונים בזמן אמת (Dieu, Lerat, & Cremer, 2023).

יתרון נוסף של WCS הוא התמיכה בעיבוד מקבילי, המאפשר פיצול שאילתות מורכבות בין שרתים מרובים לצורך ביצוע מהיר יותר. תכונה זו תומכת ביכולת להתרחב בקנה מידה גדול ללא ירידה משמעותית בביצועים, דבר חשוב במיוחד בסביבות דינמיות שבהן יש צורך להתמודד עם כמויות גדולות של נתונים בזמן אמת (Bruckner, 2012).

למרות היתרונות, Wide-Column Stores סובלים מכמה חסרונות משמעותיים. ראשית, פונקציות מתקדמות כמו חיבור (JOIN) בין טבלאות או פונקציות אגרגציה מבוצעות באיטיות רבה או אינן נתמכות כלל, מה שמהווה מגבלה משמעותית כאשר יש צורך בניתוחים מורכבים יותר של הנתונים. הדבר נובע מהעובדה שכל שאילתה כזאת דורשת שליפת נתונים ממספר שרתים בו זמנית, תהליך שיכול להיות יקר מבחינת משאבים וזמן (Dieu et al., 2023).

בנוסף, ישנם אתגרים בעקביות הנתונים. מערכות WCS, כגון Cassandra, מאמצות גישה שמעדיפה זמינות וביצועים על פני עקביות מיידית של הנתונים (מבנה CAP). משמעות הדבר היא שבחלק מהמקרים, עקביות הנתונים בין השרתים אינה מובטחת עד שתהליך תחזוקה מתבצע, מה שעלול להוביל לאי-ודאות בזמן אמת ולביצועים לא עקביים (Bruckner, 2012).

עוד חסרון קשור לצריכת משאבים: מערכות WCS רבות, כולל Cassandra, נכתבות בשפות תכנות כמו JAVA, שהן פחות יעילות מבחינת ניהול זיכרון וצריכת משאבים בהשוואה לשפות כמו C או C++. כתוצאה מכך, יש צורך במשאבים רבים יותר לתמיכה במערכות אלו, מה שיכול להוות חסם ליישום בסביבות שבהן חשובים יעילות אנרגטית והוצאות תחזוקה נמוכות (Dieu et al., 2023).

מחקרים חדשים בתחום מנסים לשלב בין היתרונות של Wide-Column Stores לבין מסדי נתונים יחסיים (RDBMS) ליצירת פתרונות היברידיים המשלבים את הגמישות והביצועים הגבוהים של WCS עם הפונקציונליות המתקדמת והעקביות של RDBMS. גישה זו נקראת Relational Wide Column Store (RWCS), והיא נועדה להציע פתרון שמותאם לדרישות של יישומים מתקדמים המחייבים גם גמישות וגם פונקציות מורכבות לעיבוד נתונים (Dieu et al., 2023).

ניתן לסכם ש Wide-Column Stores מספקים פתרונות גמישים וסקלאביליים לאחסון ועיבוד נתונים גדולים בסביבות מבוזרות, אך הם דורשים ידע טכני מתקדם לניהול ואופטימיזציה. היתרונות שלהם כוללים ביצועים גבוהים, זמינות גבוהה וגמישות מבנית, מה שהופך אותם למתאימים במיוחד ליישומים דינמיים כמו רשתות חברתיות ומערכות תוכן. עם זאת, יש לקחת בחשבון את המגבלות שלהם בתמיכה בפונקציות מתקדמות ובתחום העקביות של הנתונים, כמו גם את הדרישה למשאבי מחשוב גבוהים יותר. מגמות חדשות בתחום מנסות לפתור את הבעיות הללו באמצעות שילוב של טכנולוגיות מסדי נתונים יחסיים ומערכות (Dieu et al., 2023) NoSQL.

## מבוססי מסמכים (Document Databases):

מאגרי נתונים מסוג מסמכים (Document Databases) מהווים סוג של מאגרי נתונים NoSQL המתמקדים בניהול, אחסון וחיפוש של נתונים לא מובנים באמצעות פורמטים גמישים כמו JSON, BSON או XML. הם מציעים יתרונות רבים שמפשטים את העבודה עם סוגים שונים של מידע, במיוחד כאשר מדובר בנתונים שאינם ניתנים לייצוג בקלות במבנים טבלאיים רלציוניים. בעשור האחרון, מסדי נתונים אלו הפכו לפופולריים בשל הגמישות והיכולת שלהם להתמודד עם נתונים דינמיים ומורכבים (Clifton & Garcia-Molina, 2000; Vera et al., 2015).

מסדי נתונים מבוססי מסמכים (Document Databases) בנויים בצורה גמישה ומבוססים על מסמכים כיחידות אחסון עיקריות. כל מסד נתונים (Database) מכיל אוספים (Collections), כאשר כל אוסף הוא קבוצה של מסמכים (Documents). המסמכים עצמם מאורגנים בפורמט של JSON, BSON, XML, וכל מסמך הוא יחידת נתונים עצמאית שמכילה שדות (Fields), המהווים צמדי מפתח-ערך. המפתח הוא שם השדה, והערך יכול להיות מחרוזת, מספר, מערך, או מסמך נוסף (Nested Document). כל מסמך מזוהה על ידי מזהה ייחודי (Unique Document ID). קיים מושג נוסף שנקרא Sub-Collection (תת-אוסף). Sub-Collection הוא אוסף מקונן בתוך מסמך (Document) קיים באוסף (Collection) עיקרי. המשמעות היא שניתן לאחסן אוספים בתוך מסמכים, כך שכל מסמך יכול להכיל אוספים משלו, דבר שמוסיף רמת היררכיה וגמישות למבנה הנתונים.

לדוגמה מבנית: מסמך עם מזהה ייחודי: user\_123

שדות:

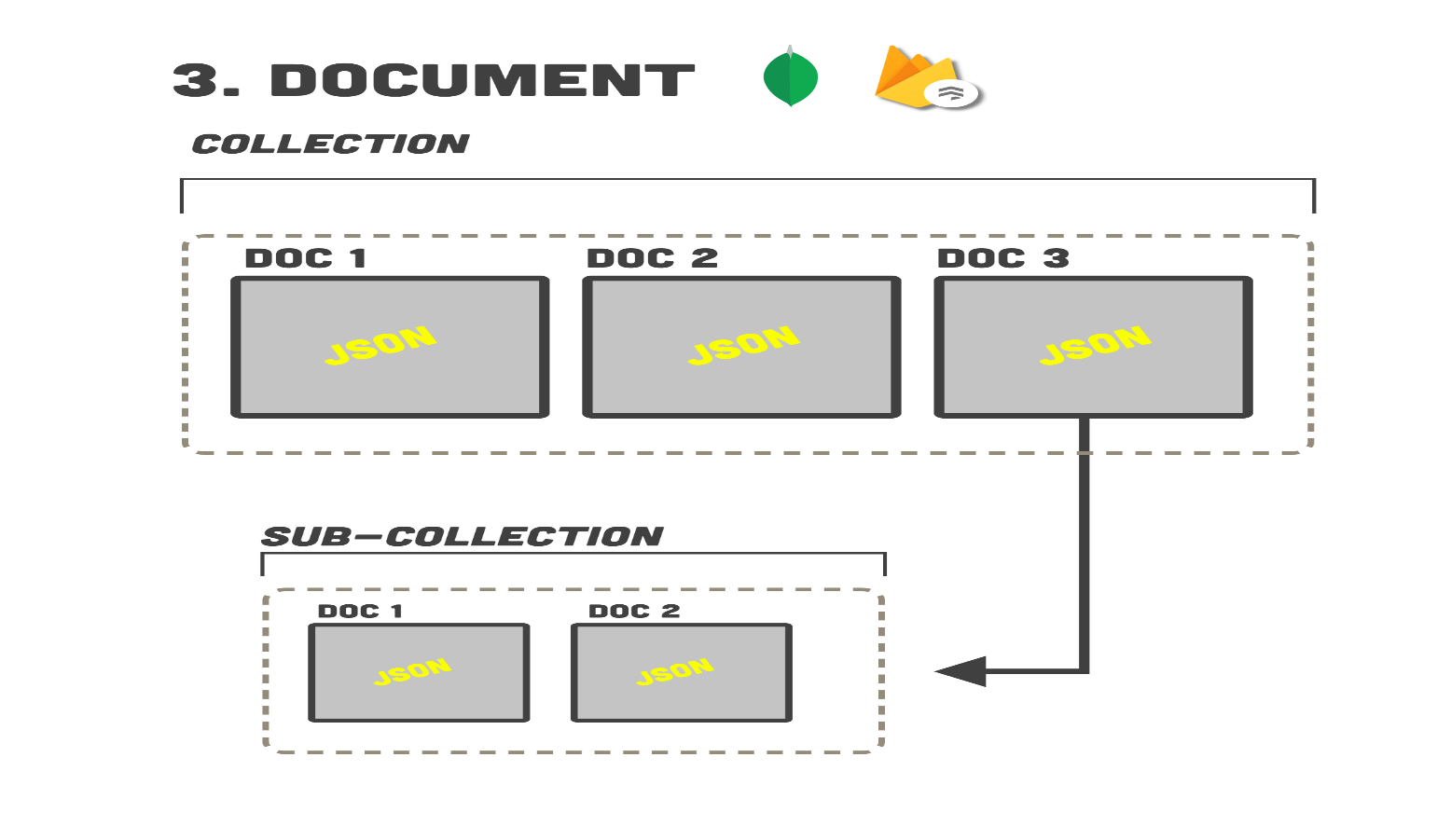
FirstName: John

LastName: Doe

תת-אוסף (Sub-Collection): Orders

מסמך (Document) בתוך תת-האוסף:

OrderID: order\_001

Date: 2024-09-25

(Fireship. (n.d.). Top seven database paradigms. Fireship.io)

אחד היתרונות המרכזיים של מסדי נתונים מסוג מסמכים הוא המבנה הגמיש של המסמכים עצמם. בניגוד למסדי נתונים רלציוניים, המארגנים נתונים בטבלאות עם שורות ועמודות, מסדי נתונים מבוססי מסמכים מאחסנים נתונים בצורה של מסמכים עצמם, כל אחד מהם יכול להכיל שדות במבנים שונים. זה מאפשר למפתחים להתאים את מבנה המסמכים לצרכים משתנים של היישום, ולהוסיף או לשנות שדות מבלי לפגוע במבנה הכללי של הנתונים. פורמטים כמו JSON ו-BSON מאפשרים גמישות רבה בעבודה עם נתונים משתנים ונתונים שאינם ניתנים לייצוג בצורה נוחה במבנים רלציוניים (Mihai, 2020).

מסדי נתונים מסוג מסמכים מציעים גם תמיכה נרחבת באחסון נתונים לא מובנים. יכולת זו חיונית במיוחד כאשר מדובר בקבצים גדולים או משתנים, כמו טקסטים, תמונות, קבצי PDF ווידאו, שהם חלק בלתי נפרד מהיישומים המודרניים. תכונה זו מקלה על עבודה עם מידע מגוונים ומשתנים, ומפשטת את האחסון והניהול של קבצים מסוגים שונים. מסדי נתונים מסוג מסמכים מאפשרים עבודה עם נתונים בצורה טבעית ויעילה, מבלי להסתמך על מבנים קשיחים של טבלאות (Vera et al., 2015).

ביצועים מהירים הם יתרון נוסף של מסדי נתונים מסוג מסמכים. הם מבוססים על עקרונות של אחסון יעיל ושליפת נתונים בצורה אופטימלית, מה שמאפשר חיפושים מורכבים וביצוע שאילתות על מסמכים בודדים או קבוצות של מסמכים במהירות רבה. זה חשוב במיוחד עבור יישומים הדורשים גישה מהירה לנתונים ויכולת לבצע שאילתות מורכבות על מידע מגוון. נוסף לכך, מסדי נתונים מסוג מסמכים תומכים בקנה מידה גבוה (scalability), מה שמאפשר להם להתמודד עם כמויות גדולות של נתונים ולעמוד בעומסים גבוהים. הם יכולים לבצע התרחבות בקלות בהתאם לצרכים של האפליקציה, מה שמבטיח ביצועים גבוהים גם בסביבות עם דרישות גדולות או משתנות (Clifton & Garcia-Molina, 2000).

אבטחת המידע היא מרכיב חשוב במסדי נתונים מסוג מסמכים. פתרונות הצפנה ובקרת גישה הם חלק מהותי במערכות אלו, וניתנים להתאמה אישית לפי הצרכים הספציפיים של הארגון או היישום. הם מבטיחים שמירה על בטיחות המידע ומגנים עליו מפני גישה לא מורשית או פרצות אבטחה (Mihai, 2020).

מסדי נתונים מסוג מסמכים גם תומכים בשיתוף פעולה בין משתמשים, כולל גרסאות והיסטוריית שינויים. תכונה זו מקלה על עבודה קבוצתית על מסמכים, מה שמועיל במיוחד במצבים שבהם מספר משתמשים עובדים על אותו מסמך או על קבוצות של מסמכים, ודרושים מעקב אחר שינויים ושיתוף מידע בצורה מסודרת (Vera et al., 2015).

השימושים הנפוצים במסדי נתונים מסוג מסמכים כוללים ניהול תוכן, מערכות ניהול מסמכים ארגוניים ויישומי אינטרנט המבוססים על תוכן דינמי. לדוגמה, הם מתאימים במיוחד לניהול תוכן דינמי כמו מאמרים, דוחות או קטעי טקסט המעדכנים באופן תדיר. בנוסף, הם מציעים פתרונות יעילים עבור מערכות ניהול מסמכים ארגוניים שבהן יש צורך בארגון ושיתוף מסמכים בתוך הארגון (Mihai, 2020).

דוגמאות לפלטפורמות פופולריות בתחום כוללות את MongoDB, CouchDB ו-Amazon DocumentDB, כל אחת מהן מציעה פתרונות שונים עם תכונות המותאמות לצרכים שונים של יישומים ותחומים (Clifton & Garcia-Molina, 2000). MongoDB, לדוגמה, מציע יכולות חיפוש מתקדמות ותמיכה בסקלאביליות גבוהה, מה שמאפשר לו לשמור על ביצועים גבוהים גם עם נתונים גדולים ומורכבים. CouchDB מתמקד בניהול מסמכים עם תמיכה בהיסטוריית שינויים ושיתוף פעולה, בעוד ש-Amazon DocumentDB מציע פתרון מנוהל שמתאים לשימושים בארכיטקטורות ענן, מה שמקל על ניהול התשתיות ושיפור הביצועים (Vera et al., 2015).

## מבוססי גרפים (Graph Stores):

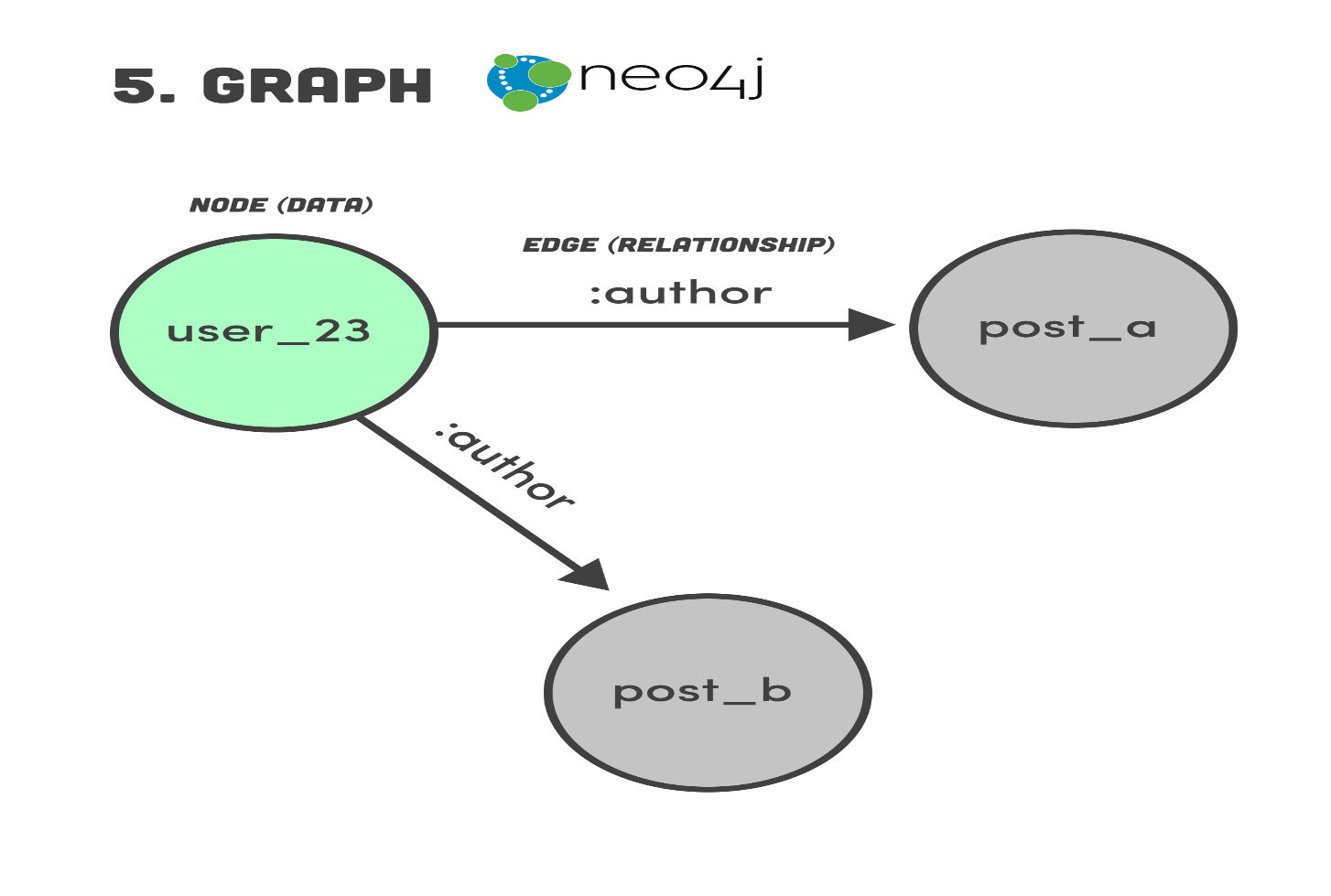
מאגרי נתונים גרפיים (Graph Stores) הם סוג של מאגרי נתונים NoSQL המתמקדים בניהול ותחזוקה של נתונים בצורה של גרפים. הם מיועדים לאחסן מידע בצורה של צמתים (nodes), קשרים (relationships) ותכונות (properties) שמספקות מידע נוסף על הצמתים והקשרים. צמתים מייצגים ישויות או אובייקטים בתוך הגרף. כל צומת יכול להכיל תכונות שמספקות מידע נוסף עליו, כמו שמות, גיל, או כתובת במקרה של צומת המייצגת אדם. קשרים מייצגים את הקשרים בין הצמתים, והם מכילים מידע על סוג הקשר ואת כיוונו. לדוגמה, קשר "חבר של" בין שני צמתים יכול לכלול תכונות כמו תאריך ההתחלה של החברות. תכונות הן זוגות מפתח-ערך שמספקות מידע נוסף על הצמתים והקשרים, מה שמאפשר יצירת מודל נתונים עשיר ומפורט. (Akid et al., 2022; Agrawal & Patel, 2016).

דוגמה מבנית:

(Neo)-[:LOVES]->(Trinity)

(Morpheus)-[:KNOWS]-(Trinity)

(user\_23)-[:author]-(post a)



(Fireship. (n.d.). Top seven database paradigms. Fireship.io).

מאגרי נתונים גרפיים מציעים מספר יתרונות חשובים. הם מתאימים במיוחד למערכות שבהן קשרים בין נתונים הם מורכבים או מרובים, לדוגמה ברשתות חברתיות, שם ניתן לנהל את הקשרים בין משתמשים, פוסטים, תגובות ותגובות לפוסטים בצורה אפקטיבית. הם מספקים ביצועים גבוהים במיוחד כאשר מדובר בשאילתות שמערבות קשרים רבים, הודות ליכולת לעבור בצורה מהירה בין הצמתים והקשרים. כמו כן, המודל הגרפי מאפשר הוספת קשרים ותכונות חדשים בקלות מבלי לשבש את המבנה הכללי של הנתונים, מה שמספק גמישות רבה בניהול הנתונים.

דוגמאות למאגרי נתונים גרפיים כוללות את Neo4j, אחד מהפופולרים ביותר, המציע ממשק משתמש נוח ושפה לשאילתות בשם Cypher, המאפשרת לבצע שאילתות בקלות ולתחקר את הגרפים בצורה אינטואיטיבית. AllegroGraph הוא מאגר נתונים גרפי המיועד ליישומים של נתונים גדולים וניתוח נתונים מורכב, עם יכולות מתקדמות לניתוח קשרים ונתונים. OrientDB משלב בין מאגרי נתונים גרפיים ומסמכים, ומספק תמיכה לשני המודלים, מה שמאפשר גמישות רבה בניהול הנתונים. Akid et al., 2022))

השימושים הנפוצים של מאגרי נתונים גרפיים כוללים ניהול רשתות חברתיות, ניהול קשרים בין משתמשים, פוסטים, תגובות ותגובות לפוסטים. הם משמשים גם במערכות ניהול תוכן, לנהל קשרים בין תכנים שונים כמו מאמרים, תגובות, קטגוריות ותוויות. ניתוח נתונים הוא תחום נוסף שבו מאגרי נתונים גרפיים נמצאים בשימוש, כמו חקירות פליליות, ניתוח רשתות, וניתוח התנהגות משתמשים.

עם זאת, קיימים אתגרים בשימוש במאגרי נתונים גרפיים. תמיכה בגרפים מקוננים היא מוגבלת ברוב מאגרי הנתונים הגרפיים, דבר שמגביל את היכולת לייצג מבנים מורכבים יותר. השפות לשאילתות במאגרי נתונים גרפיים עשויות להיות מורכבות יותר מ-SQL, מה שדורש למידה נוספת ועשוי להקשות על המפתחים שאינם מכירים את המודל הגרפי. מאגרי נתונים גרפיים עשויים להיות מורכבים יותר להבנה ולתחזוקה, במיוחד עבור צוותי פיתוח שאינם מנוסים בתחום. (Akid et al., 2022)

התחזוקה והניהול של גרפים גדולים ומורכבים עשויים לדרוש הבנה מעמיקה של המודל הגרפי. בנוסף, עבור שאילתות מסוימות, במיוחד כשמדובר בכמויות גדולות מאוד של נתונים או חישובים מורכבים, מאגרי נתונים גרפיים עשויים להיתקל בבעיות ביצועים. ייתכנו גם אתגרים בתמיכה ובסטנדרטיזציה, שכן לא כל הכלים והטכנולוגיות תומכים במאגרי נתונים גרפיים, מה שעלול להגביל את השימוש בהם בסביבות מסוימות או בדרישות מסוימות של אינטגרציה עם מערכות אחרות. אינטגרציה עם מסדי נתונים רלציוניים יכולה להוות אתגר נוסף, במיוחד כאשר יש צורך בשימוש משולב בין שני סוגי המסדים, מה שעלול להוביל לבעיות בתיאום בין המודלים ולתיאום נתונים בין מערכות שונות. (Akid et al., 2022; Agrawal & Patel, 2016).

# בסיסי נתונים מסוג NoSQL

## MongoDB:

MongoDB היא מערכת ניהול מסדי נתונים לא-יחסית (NoSQL) המתמקדת במתן פתרונות למערכות שמצריכות גמישות וסקלאביליות גבוהה בניהול ותחזוק נתונים גדולים ומורכבים. הנתונים ב-MongoDB נשמרים בפורמט BSON (Binary JSON), המאפשר יצירת מסמכים דינמיים שבהם ניתן לשלב סוגים שונים של נתונים ומבנים מקוננים. כל מסמך ב-MongoDB יכול להכיל שדות שונים, כולל מערכים ומסמכים מקוננים, מה שמספק גמישות רבה בהגדרת הנתונים. המסמכים מאוחסנים באוספים (Collections) – קבוצות של מסמכים שאין צורך להגדיר את מבניהם מראש, מה שמאפשר לעדכן ולשנות את מבנה הנתונים ללא הפרעה למסמכים אחרים באוסף (Győrödi et al., 2015; Chauhan, A., 2019).

MongoDB מציעה ביצועים גבוהים בזכות אינדוקסציה מתקדמת, המאפשרת גישה מהירה לנתונים, ומתאימה במיוחד ליישומים עם דרישות קריאה וכתיבה גבוהות כמו רשתות חברתיות, מערכות ניהול תוכן ויישומים מבוססי אינטרנט. היכולת לבצע שאילתות דינמיות ב-MongoDB, כמו גם האגרגציה המתקדמת, מאפשרת חיפושים מורכבים וניתוח נתונים בזמן אמת. MongoDB תומכת גם בהרחבה אופקית (Sharding), המאפשרת פיזור הנתונים על פני מספר שרתים בהתאם לפרמטרים מוגדרים, כגון עמודת מפתח, דבר המייעל את הביצועים ומפזר את העומס על פני מספר שרתים (Győrödi et al., 2015; Chauhan, A., 2019).

אחד היתרונות המרכזיים של MongoDB הוא היכולת לבצע שכפול נתונים באמצעות Replica Sets – קבוצות של שרתים המאגדות את הנתונים ומביאות עותקים של המידע לשרתים נוספים. תכונה זו מבטיחה את זמינות הנתונים במקרה של כשל בשרת הראשי ומספקת גיבוי אוטומטי, מה שמסייע להבטיח שהנתונים יהיו זמינים גם במקרה של כשל בשרת הראשי (Győrödi et al., 2015). יתרון נוסף הוא השכפול המאפשר גיבוי אוטומטי של הנתונים, דבר שמקטין את הסיכון לאובדן מידע.

בהתאם לצרכים הספציפיים של הפרויקט, MongoDB מתאימה במיוחד ליישומים שמחפשים גמישות וסקלאביליות, כמו ניהול Big Data, פלטפורמות ניהול תוכן, אפליקציות אינטרנטיות, אפליקציות ניידות, וניהול נתוני משתמשים. היא מספקת פתרון יעיל למערכות עם דרישות ביצועים גבוהות ונתונים לא-מובנים כמו תמונות, קבצי וידאו ומסמכים, מה שמאפשר למפתחים לעבוד עם סוגי נתונים שונים בקלות רבה. MongoDB Atlas, פתרון הענן של MongoDB, מאפשר הקמה וניהול פשוט של מסדי נתונים עם יכולות גיבוי, שדרוגים וניהול אוטומטי, מה שמקל על תחזוקה ושדרוג של המערכת (Mongo, D. B., 2015; Chauhan, A., 2019).

עם זאת, קיימים חסרונות לשימוש ב-MongoDB. כתיבה אסינכרונית עשויה להוביל לחוסר עקביות של הנתונים, והיעדר סכימה קבועה עלול לגרום לקשיים בניהול עקביות הנתונים ובשמירה על התקינות של הנתונים. אמנם MongoDB שיפרה את התמיכה בטרנזקציות, היא עדיין אינה תומכת ברמת התמיכה שמציעים מסדי נתונים יחסיים (Chauhan, A., 2019). בעת עבודה עם מסמכים גדולים מאוד, ייתכן שהביצועים ייפגעו, במיוחד אם יש צורך לעדכן מסמכים בתדירות גבוהה.

ניהול מסדי נתונים גדולים עם MongoDB עשוי להיות מורכב ודורש הבנה מעמיקה של המערכת, במיוחד כשמדובר בשידור ובביצוע אופטימיזציות. יתר על כן, בעוד שמסגרת האגרגציה חזקה, שאילתות מורכבות עשויות להיות קשות לכתיבה ולתחזוקה, במיוחד בהשוואה לשפות SQL מסורתיות.

ניתן לסכם שMongoDB היא מערכת ניהול מסדי נתונים גמישה ועוצמתית המתאימה למגוון רחב של יישומים. היא מציעה פתרונות גמישים וסקלאביליים למערכות הדורשות עיבוד נתונים בזמן אמת, ניהול נתונים לא-מובנים ונתונים בכמויות גדולות. עם זאת, יש לקחת בחשבון את היתרונות והחסרונות של המערכת בהתאם לצרכים הספציפיים של הפרויקט.

## Cassandra:

Cassandra היא מערכת ניהול מסדי נתונים פתוחה (open source) שנועדה לנהל כמויות עצומות של נתונים תוך שמירה על זמינות גבוהה, סקלאביליות ויכולת להתמודד עם כשלים תכופים. המערכת פותחה על ידי אבינש לקשמן ופרשנט מאליק בפייסבוק ופורסמה לציבור בשנת 2008. כיום היא מנוהלת על ידי Apache Software Foundation. Cassandra עוצבה כך שתתמוך בסביבות שבהן יש צורך בביצועים גבוהים וביכולת להתמודד עם נתונים מגוונים ומבוזרים (Lakshman & Malik, 2010; Anusha et al., 2021).

Cassandraפועלת על פי עקרונות של ארכיטקטורת "peer-to-peer", שבה כל צומת במערכת נחשב לשווה ואין צומת מרכזי או מאסטר. (Anusha et al., 2021). כל צומת במערכת יכול לטפל בבקשות כתיבה וקריאה, מה שמפחית את הסיכון לנקודת כשל אחת ומאפשר לצמתי המערכת להמשיך לפעול גם כאשר חלקם נכשל. ארכיטקטורה זו מספקת גמישות רבה ומאפשרת הוספת צמתים חדשים בקלות, מבלי להפסיק את הפעולה של המערכת. התהליך הזה מאפשר למערכת להתרחב דינמיקה בהתאם לצרכים המשתנים של הארגון (Lakshman & Malik, 2010).

אחד היתרונות המרכזיים של Cassandra הוא היכולת שלה להתמודד עם כשלים תכופים של רכיבים באמצעות שכפול נתונים בין מספר צמתים. Cassandra מתמודדת עם כשלים תכופים של רכיבים באמצעות שכפול נתונים בין מספר צמתים" (Lakshman & Malik, 2010). הנתונים משוכפלים בין הצמתים במערכת, כך שאם צומת אחד נכשל, הנתונים עדיין זמינים בצמתים אחרים. ניתן לקבוע את מספר השכפולים הנדרשים לפי הצורך, מה שמאפשר גמישות ברמות האמינות ויכולת להבטיח שהנתונים יישמרו וזמינים גם במקרה של כישלון צומת (Anusha et al., 2021).

Cassandra מציעה ביצועים גבוהים במיוחד בתחום הכתיבה. המערכת מתמקדת בביצועים גבוהים במיוחד בתחום הכתיבה ומסוגלת לבצע כתיבות רבות במקביל (Anusha et al., 2021). היא מסוגלת להתמודד עם עומסים כבדים ולספק זמני תגובה מהירים גם כאשר כמות הנתונים גדלה משמעותית. היכולת שלה לעבד מיליארדי כתיבות ביום היא קריטית עבור יישומים כמו פייסבוק שבהם יש צורך לעבד כמויות עצומות של נתונים בזמן אמת. זמני התגובה של המערכת נמוכים, עם מינימום של 7.69ms לחיפושי מילות מפתח, דבר שמבטיח חוויית משתמש טובה גם כאשר הכמויות גדולות מאוד (Lakshman & Malik, 2010).

Cassandra משתמשת בשפת שאילתות ייחודית בשם Cassandra Query Language (CQL), הדומה ל-SQL אך מותאמת למבנה הנתונים הקולומנרי שלה. (Anusha et al., 2021) Cassandra מאפשרת למפתחים לבצע שאילתות בצורה קלה ומוכרת, מה שמפשט את המעבר למערכת ומקל על העבודה עם נתונים מבוזרים. המערכת תומכת בשני סוגי משפחות עמודות: משפחות עמודות פשוטות ומשפחות עמודות עליונות, מה שמאפשר גמישות רבה בניהול הנתונים (Lakshman & Malik, 2010).

עם זאת, קיימים גם חסרונות בניהול Cassandra. ניהול קלאסטר של Cassandra יכול להיות מורכב, במיוחד עם מספר גדול של צמתים. (Lakshman & Malik, 2010). הצורך בהגדרה ומעקב קפדני של המערכת יכול להוות מעמסה על המנהלים, ודורש ידע טכני מעמיק כדי למקסם את הביצועים. יתרה מכך, Cassandra אינה כוללת מסגרת אגרגציה מובנית, ולכן יש צורך בכלים חיצוניים כמו Apache Spark או Hadoop לצורך אגרגציה של נתונים (Anusha et al., 2021).

המערכת גם לא מתאימה לשאילתות מורכבות כמו JOINs, מה שיכול להוות מגבלה עבור יישומים שדורשים חיבורים בין טבלאות או שאילתות רב-שכבתיות. עיצוב המערכת יכול להוביל לבעיה של הגדלת כתיבה, שבה כמות הנתונים שנכתבים לדיסק הרבה יותר מהנתונים בפועל שנשמרים. דבר זה עשוי להוביל ללבוש מוגבר על מכשירי האחסון ולביצועים מופחתים עם הזמן (Lakshman & Malik, 2010). בנוסף, למרות שהיא תוכננה לפעול על חומרה זולה, Cassandra יכולה להיות תובענית במשאבים, במיוחד מבחינת זיכרון ו-I/O דיסק, מה שעשוי לדרוש חומרה יותר חזקה לביצועים אופטימליים (Anusha et al., 2021).

Cassandra נמצאת בשימוש נרחב על ידי חברות טכנולוגיה גדולות כמו פייסבוק, אינסטגרם, אפל ונטפליקס, שמנצלות את יתרונות המערכת כדי לנהל כמויות עצומות של נתונים בזמן אמת. היא מתאימה במיוחד ליישומים שדורשים זמינות גבוהה, כמו מערכות ניהול תוכן, רשתות חברתיות ויישומים של ניתוח נתונים בזמן אמת. באופן כללי, Cassandra היא פתרון מצוין עבור ארגונים שזקוקים למערכת ניהול נתונים חזקה, גמישה ומבוזרת, במיוחד כאשר מדובר בכמויות גדולות של נתונים ובדרישות גבוהות של זמינות וביצועים (Anusha et al., 2021).

## Redis:

Redis היא מערכת ניהול מסדי נתונים NoSQL פתוחה המתמקדת באחסון נתונים במבנה של מפתחות-ערכים . (key-value store) Redis נחשבת לאחת מהפתרונות המובילים בתחום בזכות ביצועיה הגבוהים, גמישותה הרבה וקלות השימוש שבה (Seghier & Kazar, 2021). המערכת מתבססת על זיכרון RAM, מה שמאפשר לה לספק ביצועים יוצאי דופן עם זמני תגובה נמוכים מאוד ועיבוד מיליוני בקשות בשנייה (Puangsaijai & Puntheeranurak, 2017).

Redis מציעה מגוון רחב של מבני נתונים, כולל מפתחות (Strings), רשימות (Lists), קבוצות (Sets), מפות (Hashes), קבוצות מסודרות (Sorted Sets) ו-Streams. הגמישות של Redis טמונה בכך שהיא אינה דורשת הגדרת סכימה קבועה, מה שמאפשר אחסון נתונים במבנים שונים ובצורות שונות. המערכת תומכת בעסקאות (transactions) שמבצעות מספר פקודות כקבוצה אחת, מה שמבטיח שכל הפקודות יתבצעו בהצלחה או לא יתבצעו כלל (Puangsaijai & Puntheeranurak, 2017).

Redis מציעה מספר תכונות מתקדמות נוספות, כולל תמיכה בשכפול נתונים (replication) ופיזור נתונים (sharding). מודל השכפול מאסטר-עבד מאפשר לשכפל נתונים ממאסטר אחד למספר עבדות, מה שמסייע בגיבוי נתונים ובשיפור ביצועים על ידי פיזור העומס בין מספר עבדות. הפיזור בין מספר צמתים מאפשר שיפור בזמינות הנתונים ומפזר את העומס על מספר שרתים, דבר שמגביר את הסקלביליות של המערכת. Redis תומכת גם במודלים של peer-to-peer, המציעים גמישות נוספת בניהול הנתונים (Almeida et al., 2023).

Redis כוללת יכולות סקריפטינג באמצעות Lua, המאפשרות להריץ קוד ישירות על השרת. זה מפחית את הצורך בהעברת נתונים בין השרת ללקוח ומייעל את הביצועים (Almeida et al., 2023). המערכת מציעה גם תכונות כמו Redis Sentinel, המאפשרות ניהול אוטומטי של כישלונות והפיכת עבדות למאסטר במקרה של כשל במאסטר, ו-Redis Cluster, המאפשר פיזור נתונים בין מספר צמתים ליכולת סקלביליות מוגברת (Seghier & Kazar, 2021).

למרות יתרונותיה הרבים, Redis אינה חפה מחסרונות. מאחר שהיא מאחסנת את כל הנתונים בזיכרון, זה עלול להיות יקר עבור כמויות גדולות מאוד של נתונים. יתרה מכך, Redis אינה תומכת בשאילתות SQL מורכבות כמו מסדי נתונים רלציוניים, דבר שעשוי להגביל את השימוש בה במקרים מסוימים (Puangsaijai & Puntheeranurak, 2017). גם ניהול נתונים גדולים מאוד יכול להיות מאתגר, ואם לא מוגדרים פתרונות גיבוי מתאימים, הנתונים עלולים להיאבד במקרה של קריסת המערכת (Almeida et al., 2023).

Redis נהנית מקהילה פעילה מאוד עם תיעוד נרחב, מדריכים ותוספים המאפשרים הרחבה של הפונקציות הבסיסיות שלה, מה שמקל על מציאת משאבים ותמיכה (Seghier & Kazar, 2021). בסיכומו של דבר, Redis היא פתרון מצוין עבור יישומים שדורשים ביצועים גבוהים וגמישות בניהול נתונים, אך חשוב לקחת בחשבון את הצרכים הספציפיים של היישום ואת העלויות הנלוות לשימוש בזיכרון.

## Neo4J:

Neo4j היא מערכת ניהול מסדי נתונים גרפיים המיועדת לניהול, אחסון ועיבוד נתונים בצורה של גרפים, המייצגים קשרים מורכבים בין נתונים. בניגוד למסדי נתונים רלציוניים המשתמשים בטבלאות כדי לנהל את הנתונים, Neo4j מבוססת על מבנה גרפי שבו כל רכיב נתונים מיוצג כצומת (node), והקשרים ביניהם מיוצגים כקווים מקשרים(relationships) (Miller, J. J. 2013). כל צומת יכול לייצג אובייקט כמו אדם, מוצר או אירוע, והקשרים בין הצמתים מייצגים את האינטראקציות ביניהם, כמו "חבר של", "קנה", או "השתתף ב" (Sholichah et al., 2020).

Neo4j מציעה יתרונות משמעותיים בניתוח קשרים בין נתונים, הודות לתכנון המאפשר ביצוע שאילתות על גרפים בצורה מהירה (Miller, J. J. 2013). מערכת זו מתמקדת בקשרים ובתובנות הנובעות מהם, ומאפשרת ביצועים גבוהים במיוחד בשאילתות על קשרים. ניתוח כזה מאפשר לחשוף תובנות משמעותיות אודות הקשרים בין נתונים, במיוחד כאשר מדובר במערכות גדולות ורב-שכבתיות (Sholichah et al., 2020).

שפת השאילתות Cypher, שנועדה במיוחד עבור Neo4j, מאפשרת למפתחים ולמשתמשים לבצע חיפושים ואנליזות על הגרפים בצורה אינטואיטיבית וברורה (Miller, J. J. 2013). Cypher נועדה להקל על כתיבת שאילתות מורכבות ולבצע חיפושים על הגרפים באופן נגיש, מה שמפשט את השימוש במערכת גם למי שאינו מומחה בתחום הגרפים. היא מאפשרת ביצוע חיפושים מורכבים, חישוב מסלולים, וניתוח נתונים בצורה ברורה ויעילה (Sholichah et al., 2020).

יתרון נוסף של Neo4j הוא גמישות בניהול והוספת נתונים. המערכת אינה דורשת הגדרת סכמות קשיחות מראש, ולכן ניתן להוסיף סוגי נתונים חדשים ולבצע שינויים במבנה הנתונים לאורך זמן, מבלי להיתקל בקשיים המאפיינים מערכות רלציוניות (Miller, J. J. 2013). גמישות זו מתאימה במיוחד למצבים שבהם יש צורך להוסיף נתונים חדשים או לשנות את מבנה הנתונים בצורה תדירה. Neo4j מציעה תמיכה בנתונים לא מובנים, מה שהופך אותה לאידיאלית עבור יישומים כמו רשתות חברתיות, ניתוח קשרים ומערכות המלצה (Sholichah et al., 2020).

יתרה מכך, המערכת כוללת מנגנונים מתקדמים לאופטימיזציה של שאילתות, המאפשרים ביצוע יעיל של שאילתות על פני גרפים גדולים. לדוגמה, בתחום רשתות חברתיות, Neo4j יכולה לנתח קשרים בין משתמשים, לזהות קבוצות או "קהילות" בתוך הרשת, ולעקוב אחר התפשטות של מידע או השפעה (Miller, J. J. 2013). בתחום מערכות ההמלצה, Neo4j יכולה לנתח קשרים בין מוצרים ולספק המלצות מותאמות אישית על סמך הקשרים בין נתונים (Sholichah et al., 2020).

במגוון רחב של יישומים נוספים, Neo4j מציעה יתרונות כגון ניהול נתונים רפואיים, זיהוי הונאות, וניהול מערכות לוגיסטיות. לדוגמה, בתחום הבריאות, Neo4j יכולה לשמש לניתוח קשרים בין חולים, רופאים ומחלות, מה שמאפשר גילוי תובנות על בסיס הקשרים בין גורמים שונים (Miller, J. J. 2013). בתחום זיהוי ההונאות, Neo4j יכולה לנתח דפוסי התנהגות וקשרים בין חשבונות, עסקאות ואנשים כדי לזהות התנהגות חריגה או חשודה (Sholichah et al., 2020).

עם זאת, ישנם גם חסרונות לשימוש ב-Neo4j. המסד נתונים דורש יותר זיכרון לעומת מסדי נתונים רלציוניים כמו MySQL, מה שעלול להיות בעייתי בסביבות עם משאבים מוגבלים (Miller, J. J. 2013). בנוסף, למפתחים חדשים עשויה להיות עקומת למידה גבוהה יותר בהשוואה לשפות SQL מסורתיות, במיוחד אם הם אינם מכירים את עקרונות הגרפים. גם קיימת עלות גבוהה בגרסאות מסוימות, מגבלות בקנה מידה במקרים מסוימים, וחסר התאמה במודלים רלציוניים קלאסיים (Sholichah et al., 2020). כמו כן, שאילתות מורכבות מאוד עשויות להיות קשות ליישום ב-Cypher, מה שעלול להגביל את הגמישות של המערכת (Miller, J. J. 2013).

לסיכום, Neo4j הוא פתרון חזק ומתקדם לניהול נתונים גרפיים, המציע יתרונות רבים בניתוח קשרים ובגמישות בניהול נתונים לא מובנים. עם זאת, יש לקחת בחשבון את דרישות הזיכרון, עקומת הלמידה והעלויות של המערכת. השימושים הרבים של Neo4j בתחום הרשתות החברתיות, מערכות ההמלצה וניתוח נתוני בריאות מדגישים את הפוטנציאל שלו ככלי עזר חשוב בקבלת החלטות ובניתוח נתונים (Sholichah et al., 2020; Miller, J. J. 2013).

# פתרונות חיפוש וייצוג מידע

## Elastic Search:

Elasticsearch הוא מנוע חיפוש וניתוח נתונים חזק המיועד להתמודד עם סוגים שונים של נתונים, כולל נתונים מובנים ולא מובנים. מדובר במנוע חיפוש פתוח המציע יכולות חיפוש בזמן אמת ובחיפוש טקסט מלא, תוך שימוש ב-API מבוסס REST. נכתב בשפת Java, הוא נועד להיות מפוזר, סקלאבילי, ומספק חיפוש כמעט בזמן אמת (Kononenko et al., 2014).

Elasticsearch מציע תקשורת פשוטה באמצעות API RESTful, שמאפשרת תקשורת נוחה עם השרת דרך בקשות HTTP ואובייקטים בפורמט JSON, ותומכת בשפות תכנות רבות. המיפוי ב-Elasticsearch דומה להגדרת סכימה בבסיסי נתונים SQL ומאפשר הגדרה מדויקת של סוגי המסמכים ואופן אחסונם, ניתוחם ואינדוקס שלהם, עם אפשרות למיפוי מפורש לשליטה טובה יותר (Kononenko et al., 2014).

המערכת מציעה יכולות חיפוש טקסט מלא מתקדמות, ביצועים גבוהים עם זמני תגובה נמוכים, וזמינות גבוהה באמצעות ארכיטקטורת קלאסטרים המבטיחה שכפול של shards ראשיים ל-shards משניים, כך שאם נוד נופל, המידע עדיין נגיש (Kathare, Reddy, & Prabhu, 2020). בנוסף, המנוע מציע סקלאביליות גבוהה, מאפשר להרחיב את המערכת בקלות על ידי הוספת נודים נוספים, ותומך באבטחה עם מנגנונים כמו בקרת גישה מבוססת תפקידים והצפנת תקשורת (Kathare, Reddy, & Prabhu, 2020).

אחת התכונות המרכזיות של Elasticsearch היא אינדקס הפוך, המאפשר חיפושים מהירים ויעילים על ידי שמירה של מיפוי מהערכים המובילים למסמכים בהם הם מופיעים (Kononenko et al., 2014). זה מאפשר חיפושים מהירים אפילו בכמויות נתונים גדולות מאוד. המיפוי ב-Elasticsearch מאפשר גם את הגדרת מבנה המסמכים ואופן האינדוקס שלהם, מה שמייעל את יכולת החיפוש ואת ביצועי המנוע (Kathare, Reddy, & Prabhu, 2020).

המערכת תומכת גם בניתוח נתונים מתקדם, כולל חישובי סיכום וגרפים, ותומכת בנתונים גיאוגרפיים לחיפושים מבוססי מיקום (Kathare, Reddy, & Prabhu, 2020). יתרון נוסף הוא תמיכה בעיבוד מסמכים מקוננים, מה שמאפשר חיפושים וניתוחים מדויקים גם כאשר הנתונים אינם שטוחים אלא מכילים היררכיות ומבנים מורכבים (Kononenko et al., 2014).

עם זאת, ישנם גם חסרונות ושיקולים חשובים. הגדרת Elasticsearch עשויה להיות מורכבת ודורשת ידע טכני מתקדם, במיוחד בנוגע לניהול קלאסטרים, שדורש תחזוקה מתמדת (Kathare, Reddy, & Prabhu, 2020). כמו כן, חוסר האבטחה הבסיסי, כמו אימות ושליטה על גישה, דורש הגנה נוספת כמו פרוקסי או חומת אש. כתיבת שאילתות מורכבות עשויה להיות מאתגרת, במיוחד כשמדובר באובייקטים מקוננים, וחסרות תכונות כמו טרנזקציות ופעולות JOIN שנמצאות בדרך כלל בבסיסי נתונים SQL (Kononenko et al., 2014).

שימושים נפוצים ב-Elasticsearch כוללים שיפור חוויית החיפוש באתרי אינטרנט, ניתוח נתונים ממקורות שונים כמו רשתות חברתיות ויומני שרתים, ושיפור תהליכי קבלת החלטות עסקיות על ידי ניתוח נתונים בזמן אמת. לדוגמה, הכלי DASH, שנועד לעזור למפתחים לעקוב אחרי תהליכי עבודה במערכות ניהול בעיות כמו Bugzilla, השתמש ב-Elasticsearch לשיפור ביצועי החיפוש והתגובה (Kononenko et al., 2014).

בסיכום, Elasticsearch הוא כלי רב עוצמה המציע יתרונות רבים בתחום החיפוש והניתוח, אך יש לקחת בחשבון גם את האתגרים הכרוכים בשימוש בו (Kathare, Reddy, & Prabhu, 2020; Kononenko et al., 2014).

## RDF:

RDF (Resource Description Framework) הוא מודל נתונים שפותח על ידי W3C (World Wide Web Consortium) כדי לייצג מידע על משאבים באינטרנט בצורה סטנדרטית, גמישה ומבנית. RDF מבוסס על העיקרון של תיאור משאבים באמצעות טריפלטים, שהם יחידות מידע הכוללות שלושה מרכיבים עיקריים: סובייקט (Subject), פרופרטי (Predicate) ואובייקט (Object). הסובייקט הוא המשאב או הנושא של התיאור, הפרופרטי הוא המאפיין או הקשר של הסובייקט, והאובייקט הוא הערך או המשאב שאליו מתייחס הפרופרטי. לדוגמה, טריפלט טיפוסי יכול להיות "הספר - שם - ההיסטוריה של הזמן" (Angles, Thakkar, & Tomaszuk, 2020).

היכולת לייצג מידע בצורה של טריפלטים מאפשרת גמישות גבוהה בייצוג נתונים, והמידע נשמר כגרפים. כל טריפלט מהווה קשת בין שני קודקודים בגרף (הסובייקט והאובייקט), כאשר הפרופרטי מהווה את הקשת עצמה. זה מאפשר ייצוג של קשרים מורכבים בין משאבים, ומקנה אפשרות ליצירת רשתות מידע עשירות המחברות בין נתונים שונים באינטרנט (Lausen, Meier, & Schmidt, 2008). RDF

משתמש ב-URI (Uniform Resource Identifier) או IRI (Internationalized Resource Identifier) כדי לייצג משאבים. URI מספק זיהוי ייחודי של כל משאב באינטרנט, מה שמאפשר הבחנה בין משאבים שונים ויוצר קישוריות בין משאבים שונים. השימוש ב-URI מאפשר ל-RDF לתמוך בקישוריות גבוהה בין משאבים וליצור רשתות מידע מקשרות (Angles, Thakkar, & Tomaszuk, 2020).

RDF תומך בשפות תיאור נוספות כמו RDF Schema (RDFS) ו-OWL (Web Ontology Language). ,RDF Schema מוסיף סמנטיקה למונחים וליחסים בין משאבים ומספק מבנים נוספים כמו היררכיות של כיתות ותכונות. OWL מספקת רמות נוספות של סמנטיקה ומאפשרת יצירת אונטולוגיות מורכבות, הגדרות מתקדמות של הקשרים ויחסים הכרחיים, ומגבלות על הקשרים (Lausen, Meier, & Schmidt, 2008).

RDF תומך בשאילתות באמצעות SPARQL (SPARQL Protocol and RDF Query Language), שפת השאילתות הסטנדרטית עבור RDF. SPARQL מאפשרת לבצע שאילתות מורכבות על נתוני RDF, לשלוף, לעדכן ולנהל מידע מתוך גרפי RDF. עם זאת, ביצועים של שאילתות SPARQL עשויים להיחלש כאשר מדובר בכמויות גדולות של נתונים, דבר שעשוי להשפיע על היעילות של חיפושים ומניפולציות מידע (Angles, Thakkar, & Tomaszuk, 2020).

היתרונות של RDF כוללים גמישות בייצוג נתונים, קלות בשיתוף מידע בין מערכות שונות, ותמיכה בסמנטיקה המאפשרת הבנה מעמיקה של הקשרים בין נתונים. RDF מקדם קישוריות בין נתונים, מה שמאפשר יצירת רשתות מידע שמחברות בין נתונים שונים באופן אינטואיטיבי. יתרונות נוספים כוללים היכולת לייצג קשרים מורכבים בין משאבים בצורה ברורה (Lausen, Meier, & Schmidt, 2008).

עם זאת, RDF נתקל בכמה חסרונות. יישום RDF עשוי להיות מורכב ודורש הבנה מעמיקה של המודל והסטנדרטים השונים. ביצועים של שאילתות SPARQL עשויים להיחלש עם כמויות גדולות של נתונים, וחוסר אחידות בשל מספר סטנדרטים ושפות תיאור יכול לגרום לבלבול ולחוסר אחידות. בנוסף, יישום RDF עשוי לדרוש משאבים טכניים נוספים, כמו שרתים ותוכנות מתאימות, מה שעשוי להוות אתגר נוסף עבור ארגונים קטנים (Angles, Thakkar, & Tomaszuk, 2020).

באופן כללי, RDF מהווה כלי חשוב לייצוג מידע באינטרנט, עם יתרונות ברורים בגמישות וביכולת לייצג קשרים מורכבים בין משאבים. עם זאת, יש לקחת בחשבון את המורכבות ואת האתגרים הכרוכים בשימוש בו.

# השוואה בין NoSQL ל – SQL

## יתרונות וחסרונות SQL:

שפת שאילתות מובנית (SQL) מהווה אבן יסוד בתחום מסדי הנתונים היחסיים, ומשמשת ככלי אוניברסלי לניהול ולעיבוד נתונים. האימוץ הנרחב שלה מעיד על היעילות והגמישות שהיא מציעה, כאשר היא מספקת שלל יתרונות שמייעלים את תהליכי העבודה עם נתונים עבור עסקים, מפתחים ואנליסטים. עם זאת, כמו כל טכנולוגיה, גם SQL אינה חפה מחסרונות. בפרק יעסוק ביתרונות ובחסרונות של SQL, יחשוף את תפקידה המרכזי בניהול נתונים מודרני ובמקביל יתייחס לאתגרים שטכנולוגיה זו מציבה. Silberschatz et al. 2010))

לפי (2023)Mehta, D , ב- SQL תהליך Injection גורם לבעיות אבטחה וכך הוא משפיע על רוב התוכניות המבוססות על סוגים שונים של מסדי נתונים. על פי המחקר, נמצא כי Injection ב- SQL פגיע בעיקר בתוכנות הנשענות על בסיסי נתונים. החסרונות של המודלים הקיימים לזיהויSQL Injection כוללים את חוסר היכולת שלהם לזהות דפוסים חדשים. המודלים שהוזכרו במחקר זיהו רק את מה שהם כבר ראו או אומנו עליו.

ניתן למנות את יתרונות השימוש ב SQL על ידי מספר מאפיינים עיקריים:

עיבוד שאילתות מהיר ויעיל ב- SQL הפועלת במהירות ומשפרת משמעותית את תהליך שליפת הנתונים. זריזות זו חלה גם על משימות כמו שינוי נתונים ואחסונם. בהשוואה למסדי נתונים לא-מובנים כמו MongoDB, העיצוב היחסי של SQL מבטיח אחסון נתונים מסודר, החל מהשליפות ועד לחישובים אנליטיים. SQL מבצעת את המשימות במהירות גבוהה, אך יחד עם זאת, יש לקחת בחשבון שגם אם SQL מצטיינת ביעילות בעבודה עם נתונים מובנים, קיימים בה אתגרים כמו (scalability) וניהול נתונים היררכי (Silberschatz et al. (2010)

אחד היתרונות של SQL הוא היכולת לשלוף נתונים בצורה פשוטה, ללא צורך בכתיבת קוד מורכב. בניגוד לשפות תיכנות אחרות, SQL לא דורשת כישורי תיכנות מתקדמים, מה שמקל על תחזוקת בסיסי הנתונים. הפשטות הזו מתבטאת גם בשליפת הנתונים, כאשר הפקודות של SQL הדומות לאנגלית, מקלות על ההבנה והשימוש. הכללים הסינטקטיים הפשוטים של השפה תורמים לידידותיות שלה, מה שהופך את SQL לבחירה נגישה עבור לומדים ואנשי מקצוע כאחד.

הניידות היא תכונה מבוקשת בתוכנות אחסון נתונים. היא משתלבת בקלות במגוון פלטפורמות מחשבים אישיים, שרתים, טאבלטים, מחשבים ניידים ואפילו מכשירים ניידים כמו Windows, Linux ו-Mac. SQL מתאימה לשילוב בתוכנות אחרות והיא גמישה במערכות מקומיות ובאינטרנט. הנתונים ב SQL -יכולים לעבור בקלות בין מכשירים, מה שמבטיח נגישות ורציפות פשוטה. Garcia-Molina et al. 2008))

SQL - כשפת השאילתות למסדי נתונים יחסיים, זוכה להכרה עולמית, והיא מאושרת על ידי ISO וANSI- . יש לה תיעוד נרחב והיסטוריה מבוססת שמציעה למשתמשים פלטפורמה אחידה ומקובלת בכל מקום.

SQL מטפלת במהירות בשאילתות מורכבות כאשר היא מתקשרת עם מסדי נתונים. הפקודות האינטואיטיביות שלה עונות על צרכים מגוונים, מה שהופך אותה לידידותית גם עבור אנשים ללא רקע תיכנותי. השימוש בSQL - מאפשר יצירה וניהול של מסדי נתונים גדולים, שיתוף נתונים, עדכונים ושליפות ממספר טבלאות. הגמישות שלה מקדמת את השימוש בה בתחומים כמו פיתוח אתרים, ניתוח נתונים ותמיכה טכנית, ומדגישה את חשיבותה ביישומים ארגוניים. Garcia-Molina et al. 2008))

SQLמאפשרת לכל משתמש ליצור תצוגה מותאמת אישית של הנתונים. תצוגות אלו מאוחסנות בשאילתות SQL בטבלאות מסד הנתונים. התצוגות יכולות להיווצר מטבלאות יחידות או מרובות וניתן לעצב אותן בהתאם לצרכים או לתוצאות רצויות. המשתמשים יכולים לגשת לפרשנויות ייחודיות של מבנה ותוכן מסד הנתונים, וכך להתאים את חוויית העבודה עם הנתונים לצורכיהם ולהעשיר את האינטראקציה שלהם עם המידע. Silberschatz et al. 2010, Garcia-Molina et al. 2008).)

הארכיטקטורה של SQL משתלבת בצורה חלקה עם מערכות אינטרנט תלת-שכבתיות, הכוללות את רכיבי הלקוח, שרת היישומים ומסד הנתונים. היא קיימת קרוב לחמישים שנה, מה שהופך אותה לבשלה ומבוססת מאוד. לקהילה שלה יש אנשי מקצוע מנוסים רבים שמסוגלים להציע תמיכה ולהנחיל פרקטיקות מיטביות. קיימות הזדמנויות רבות לשיפור מיומנויות ולעבודה שיתופית. אם יש צורך, ניתן להיעזר ביועצים ובספקי SQL שמציעים תמיכה נוספת. Silberschatz et al. 2010, Garcia-Molina et al. 2008).)

כמו לכל פלטפורמת IT גם ל SQL ישנן חסרונות כדלקמן:

הרחבת מסדי נתונים ב SQL -דורשת השקעה במשאבים. הרחבה אנכית, המתבצעת באמצעות הוספת חומרה חזקה יותר, היא תהליך יקר וגוזל זמן. אפשרות אחרת היא הרחבה אופקית באמצעות חלוקה (partitioning), אך היא מוסיפה מורכבויות נוספות ודורשת זמן ומשאבים נוספים. טיפול במסדי נתונים גדולים מצריך לעיתים קידוד ומפתחים מיומנים, מה שמעלה את העלויות. כאשר הנתונים גדלים, הרחבת מסדי נתונים ב SQL -הופכת לתהליך מתמשך של מרדף אחר התצורה האידיאלית, שתמיד נראית קשה להשגה(Silberschatz et al. (2010).

מתכנתים העובדים עם SQL עשויים לגלות כי אין להם שליטה מלאה על מסדי הנתונים. מגבלה זו נובעת לעיתים קרובות מהנחיות ותקנות מוסדיות או תאגידיות מוסתרות שלא ניתן לעקוף בקלות.

גרסאות מסוימות של SQL עלולות להיות יקרות, מה שמונע ממתכנתים גישה אליהן. מסד נתונים ב-SQL מחייב סכמת נתונים מוגדרת מראש לשימוש. לאחר שהסכמות הללו מוגדרות, הן נותרות קשיחות, מה שהופך את השינויים בהן למורכבים וגוזלי זמן. כתוצאה מכך, נדרש מאמץ משמעותי בהגדרת המסד עוד לפני שניתן להתחיל לעבוד איתו.

מסדי נתונים בSQL - הם היעילים ביותר כאשר הנתונים הם מובנים, ואין צפי לשינויים משמעותיים בנפח או בסוג הנתונים. הם מתאימים למצבים בהם הנתונים נותרו יציבים יחסית.

מסדי נתונים יחסיים נוצרו בתקופה שבה אחסון נתונים היה יקר, במטרה לצמצם את הכפילויות בנתונים. כל טבלה מחזיקה נתונים שונים שניתן לחבר ולבצע עליהם שאילתות דרך ערכים משותפים. עם זאת, ככל שמסדי הנתונים ב SQL -גדלים, כמות ההצטרפויות (JOIN) בין טבלאות רבות עלולה להוביל לאתגרים בביצועים ולהאטות משמעותיות. Silberschatz et al. 2010))

הגישה הנפוצה להרחבת מסדי נתונים ב SQL -היא הרחבה אנכית, כלומר הוספת רכיבי חומרה כגון זיכרון ,RAM מעבד CPU או אחסון SSD לשרת הנוכחי, או מעבר לשרת חזק יותר, מה שכרוך בעלויות גבוהות. ככל שהנתונים גדלים ונדרש אחסון רב יותר וציוד מהיר יותר להתמודדות עם טכנולוגיות מתקדמות, שדרוגי חומרה הופכים להכרחיים. ספקי מסדי נתונים דורשים לעיתים קרובות שדרוגי חומרה כדי להריץ את הגרסאות החדשות שלהם.

עם זאת, שדרוגים תכופים עלולים להפוך במהירות ללא כדאיים, שכן כל עדכון כרוך בעלויות גבוהות ומשאבים. הדרישות התכופות לשדרוגי חומרה עלולות להוביל לתחזוקה יקרה ולעלויות תפעול שוטפות, שמצטברות לעלויות כוללות גבוהות. Silberschatz et al. 2010, Garcia-Molina et al. 2008).)

## יתרונות וחסרונות NoSQL:

מחקר של Davoudian 2017 בחן את הדרישות לאחסון ושאילתות של ביג דאטה חשפו חסרונות שונים במערכות מסדי נתונים רלציוניים מסורתיים. כתוצאה מכך, הופיע סוג חדש של נתונים משלימים ולא רלציוניים, הנקראה NoSQL . במאמר נחקרו בעיקר החלטות העיצוב של בסיסי נתונים מסוגNoSQL בהתייחס לארבעה עקרונות עיצוב לא אורתוגונליים של מערכות מסדי נתונים מבוזרים: מודל נתונים, מודל עקביות, חלוקת נתונים, ותאורמת CAP .

מסדי נתונים NoSQL (הידועים גם כמסדי נתונים לא-יחסיים) מגיעים עם יתרונות וחסרונות. בצד החיובי, הם ניתנים להרחבה בקנה מידה גדול יותר מאשר מסדי נתונים יחסיים מסורתיים ויכולים לאחסן מגוון רחב של פורמטים. בנוסף, הם קלים לשימוש, והגמישות שלהם יכולה לזרז את הפיתוח, במיוחד בסביבת מחשוב בענן. מסדי נתונים NoSQL פותחו כפתרון למגבלות המובנות בטכנולוגיית מסדי הנתונים היחסיים המסורתית.

מסדי נתונים NoSQL הפכו לפופולריים משום שסיפקו דרך פשוטה לאחסן נתונים ממקורות שונים תוך שימוש בפורמטים שונים. יש צורך בפחות (או כלל אין צורך) בטרנספורמציות כאשר הנתונים נשמרים או נשלפים מהאחסון. מגוון רחב של סוגי נתונים, בין אם לא-מובנים, מובנים או חצי-מובנים, יכולים להישמר במסד נתונים NoSQL .

מסדי נתונים NoSQL פותחו בתמיכה באינטרנט ובמחשוב ענן, מה שאפשר לפתח ארכיטקטורת scale-out. יכולת ההרחבה הושגה על ידי פיזור אחסון הנתונים ותהליכי המחשוב על פני אשכול מחשבים. הגדלת הקיבולת דרשה הוספת מחשבים נוספים לאשכול.

בנוסף, הסכמות של מספר מסדי נתונים ב NoSQL -הן גמישות למדי וניתנות לשליטה על ידי המפתחים. זה מקל על התאמת מסד הנתונים לסוגי נתונים חדשים. לכל גרסה ספציפית של מסדי נתונים ב NoSQL -יש יתרונות שונים, אך לכולן יש תכונות בסיסיות משותפות:

• תמיכה בעדכוני סכמות בקלות

• עיבוד כמויות גדולות של נתונים במהירות גבוהה

• אחסון ניתן להרחבה

• יכולת לאחסן נתונים מובנים, חצי-מובנים ולא-מובנים

תכונות אלה מעניקות למשתמשים יתרונות שלא מוצעים במסדי נתונים יחסיים.

יתרון נוסף של NoSQL הוא הידידותיות למפתחים. השימוש הנרחב והקבלה של מסדי הנתונים ב-NoSQL נובעים בעיקר ממפתחים שגילו שהם יכולים לפתח ולשנות יישומים ספציפיים בקלות יחסית באמצעות NoSQL . מסדי נתונים דוקומנטריים (Document Databases) יכולים להשתמש ב-JSON כדי להמיר נתונים לפורמט דמוי קוד, מה שמאפשר למפתחים לשלוט במבנה הנתונים בצורה קלה יותר. Tiwari, S. 2011))

מכיוון שמסדי NoSQL יכולים לאחסן נתונים בפורמט המקורי שלהם, אין צורך שמפתחים יתאימו את הנתונים לצורכי אחסון, על ידי שמירת הנתונים באופן מקורי. המשמעות היא גם שפחות יישומים נדרשים לפיתוח בעת השקת מסד נתונים חדש.

למסדי נתונים NoSQL יש גם מגבלות וחולשות משלהם. חוסר המבנה יכול להפוך את הנתונים לבלתי אמינים וקשים לארגון. תלונות נפוצות כוללות לדוגמא את המקרים הבאים:

NoSQL לא כולל חבילת SQL . SQL היא טכנולוגיה בשלה המיועדת לארגון נתונים, בעוד NoSQL לא. אנליסטים מכירים היטב את SQL ומשתמשים בו באופן קבוע כאשר NoSQL נועד במקור לחקור כמויות עצומות של נתונים בלתי מאורגנים שנאספו ממקורות מרובים.

לכל מסד נתונים NoSQL יש סכימת נתונים ייחודית משלו. במקרים מסוימים כמו MongoDB אין בכלל סכימות. במצבים אחרים כמו Elasticsearch הסכימות דינמיות, ובתכנונים מסוימים הן דומות למסדי נתונים יחסיים כמוCassandra . הבעיה היא שלכל מערכת ייחודית יש חוזקות וחולשות משלה, שיש להכיר לפני שבוחרים את מסד הנתונים NoSQL המתאים למקרה. Tiwari, S. 2011))

ACIDמייצג את ארבעת המאפיינים העיקריים שמגדירים טרנזקציה (אטומיות, עקביות, בידוד, ועמידות) ו NoSQL -לא תומך במאפיינים אלו.

היעדר מאפיינים אלה גורם לכך שהנתונים המאוחסנים במסד נתונים NoSQL פחות אמינים בהשוואה לאלה המאוחסנים במסדי נתונים ב- SQL.

היעדר JOIN בNoSQL- ליצירת קשרים בין טבלאות במסד הנתונים מקשה על שילוב נתונים בין שתי טבלאות ויותר. רוב השאילתות המורכבות ב SQL -כוללות פקודות ותנאי JOIN מסייע לשלב את הנתונים של שתי טבלאות או יותר.

## עקרונות ACID מול BASE:

ACID -Atomicity Consistency Isolation Durability הינו בסיס לאמינות במסדי נתונים. ACID הם אוסף של תכונות המגדירות את האמינות והשלמות של פעולות במסד נתונים. אלו הם עקרונות יסודיים שכל מערכת ניהול מסדי נתונים רלציונית צריכה לעמוד בהם כדי להבטיח שהנתונים יהיו עקביים, שלמים וזמינים. עקרונות ACID חשובים כיוון שהם מאשרים להבטיח שהנתונים יהיו עקביים ומדויקים וכך גם שלמים. ACID מאפשר לנהל נתונים אמנים ומייצר ביצועים גבוהים למערכות מרובות משתמשים. עקרונות ACID חיוניים בכל מערכת שבה שלמות הנתונים היא קריטית, כגון: מערכות בנקאות, מערכות מסחר אלקטרוני, מערכות ניהול מלאי ומערכות ניהול בסיסי נתונים. מסדי נתונים רלציוניים כמו MySQL, Oracle ו-SQL Server מבוססות על עקרונות ACID. ACID מגיע מפרדיגמה של מסד נתונים אחד עם ''משתמשים רבים'' וטרנסאקציה על מערכי נתונים ניתן לבצע רק אחת בו זמנית עם יכולת לשנות ערך (Chandra, 2015).

ב- Atomicity כל פעולה במסד נתונים, נקראת "טרנסאקציה", מתבצעת כאטום אחד בלתי ניתן לחלוקה. המשמעות היא שכל השינויים שבוצעו במסגרת טרנסאקציה מתבצעים בשלמותם או שלא מתבצעים כלל. הדוגמה לאטומיות העברה של כסף מחשבון אחד לשני. אם תהליך ההעברה נפסק באמצע, הכסף לא ייצא מחשבון אחד ולא ייכנס לחשבון השני.

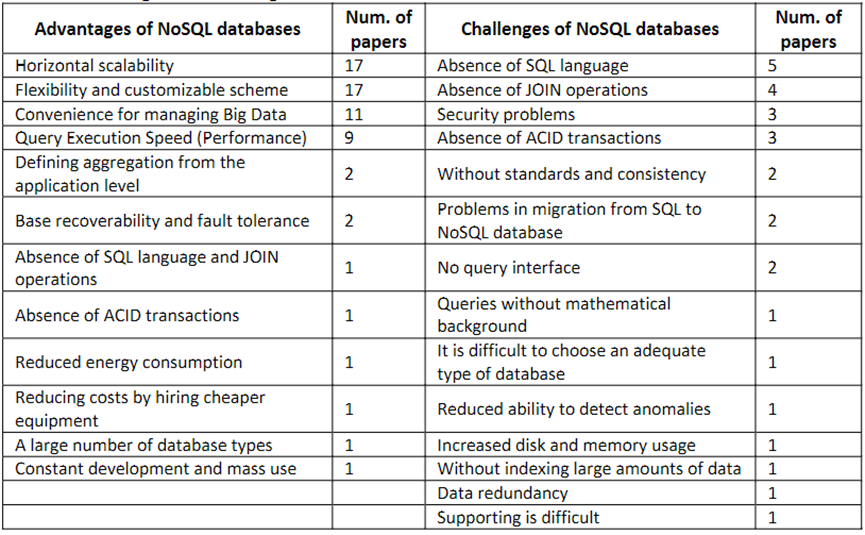
ב- Consistency כל טרנסאקציה מובילה את מסד הנתונים ממצב עקבי למצב עקבי אחר. כלומר, כללי העסקים והאילוצים על הנתונים נשמרים תמיד. אם יש כלל שאומר שמלאי מוצר לא יכול להיות שלילי, טרנסאקציה שמנסה להוציא יותר מוצרים מהמלאי תיכשל. לפיIsolation כל טרנסאקציה מתבצעת כאילו היא היחידה במערכת. כלומר, טרנסאקציות שונות לא יכולות להפריע אחת לשנייה. Durability מאפשר השלמת טרנסאקציה, השינויים שנעשו נשמרים באופן קבוע, גם במקרה של תקלה במערכת.

מנגד, BASE הוא ראשי תיבות של Basically Available, Soft state, Eventually consistent זו גישה אלטרנטיבית למסדי נתונים, המהווה מעין "אנטי-תזה" לגישתACID. בעוד ש-ACID מדגישה עקביות מיידית ושלמות נתונים, BASE מקדמת גמישות, זמינות גבוהה וסובלנות לתקלות. זמינות בסיסית המערכת תמיד זמינה, גם בזמן תקלות. תוצאות של פעולות עשויות להיות לא עקביות זמנית, אך המערכת תמשיך לתפקד. במצב רך המערכת אינה נדרשת להיות במצב עקבי באופן מיידי. שינויים עשויים להתקדם במערכת בצורה אסינכרונית. עקביות סופית המערכת תגיע למצב עקבי בנקודת זמן כלשהי בעתיד, אך אין הבטחה לגבי הזמן המדויק (Chandra, 2015). BASE

מערכות מבוזרות כמו Cassandra ו-MongoDB, העובדות על פני מספר שרתים. מדובר ביישומים הדורשים זמינות גבוהה כמו רשתות חברתיות, מערכות גיבוי. ACID ו-BASE מייצגים שתי גישות שונות לניהול נתונים. ההבדל העיקרי בין ACID ל – BASE בכך ש ACID מדגיש עקביות ושלמות הנתונים, גם במחיר של זמינות כאשר BASE מדגיש זמינות וגמישות, גם במחיר של עקביות זמנית. הבחירה בין ACID ל-BASE תלויה בדרישות היישום הספציפי. לדוגמה אם העקביות והשלמות של הנתונים הן קריטיות, כמו במערכות בנקאיות, עדיף לבחור במערכת מבוססת ACID וכאשר הזמינות הגבוהה והגמישות הן קריטיות, כמו ברשתות חברתיות, עדיף לבחור במערכת מבוסס. (Chandra, 2015) BASE

## שיקולים בבחירת NoSQL על פני SQL :

המחקר של Bozic, R. (2022) ניתח את היתרונות והאתגרים של מסדי נתונים NoSQL בהשוואה למסדי נתונים ב- SQL באמצעות סקירה שיטתית של הספרות. הניתוח במחקר בחן 40 מאמרים מדעיים. במאמר הוגדרו 12 מאפיינים עיקריים שלפיהם יש למסדי נתונים NoSQL יתרון על פני מסדי נתונים SQL ו- 14 מאפיינים בהם למסדי נתונים NoSQL יש מקום לשיפור משמעותי. להלן טבלה המרכזת את הממצאים:



ניתן לראות בסקירת הספרות שלגבי פרמטר "העדר טרנסאקציות "ACID הדעות של החוקרים חלוקות.

בעבר, מסדי נתונים SQL היו הבחירה הברורה בפיתוח יישומים חדשים כאשר בשנים האחרונות, מסדי נתונים NoSQL הפכו לחלופה פופולרית. הבחירה בין מסד נתונים SQL לעומת NoSQL הפכה להחלטה טכנולוגית חשובה עבור מפתחי צד-שרת והבנה מתי להשתמש באחד לעומת השני היא מיומנות חיונית. מעבר לכך, הבחירה בסוג מסד הנתונים יכולה לקבוע האם הפרויקט יתנהל בצורה חלקה או ייתקל בקשיים טכניים ככל שהוא יגדל.

ההבדל בין SQL לNoSQL - בא לידי ביטוי בשלמות נתונים לעומת זמינות. רוב מסדי הנתונים היחסיים מתאימים למאפייני ACID והמם מספקים הגנה לנתונים גם במקרה של שגיאות בלתי צפויות. עם זאת, זה כרוך בעלות מסוימת בביצועים, משום שהנתונים חייבים להיכתב לדיסק לפני שהלקוח מקבל תגובה מוצלחת.

לעומת זאת, מסדי נתונים NoSQL מסתמכים על מודל עקביות מתמשכת Eventual Consistency כדי להשיג זמינות גבוהה. מסדי נתונים NoSQL אינם ממתינים עד שהנתונים ייכתבו לדיסק לפני שליחת התגובה ללקוח, כפי שמוצג באיור הבא.

כתוצאה מכך, מסדי נתונים יחסיים מציעים הבטחות לשלמות הנתונים אך עשויים להגיב לאט יותר לעומת מסדי נתונים NoSQL המציעים רמה גבוהה של זמינות על חשבון רמת אמינות מסוימת.

הבדל נוסף הינו שמסדי נתונים יחסיים משתמשים בסכמות כדי לתאר את המסד ואת סוגי הנתונים שהוא מעבד. לעומת זאת, מערכות NoSQL אינן משתמשות בסכמות מסד נתונים, כך שמהיר יותר עבור המשתמש להוסיף נתונים בפורמט שונה מבלי שיצטרך לבצע שינויים נוספים במסד הנתונים. יש כאן בחירה בין נוחות השימוש של מסדי נתונים NoSQL לבין איכות ועקביות הנתונים במסדי נתונים SQL .

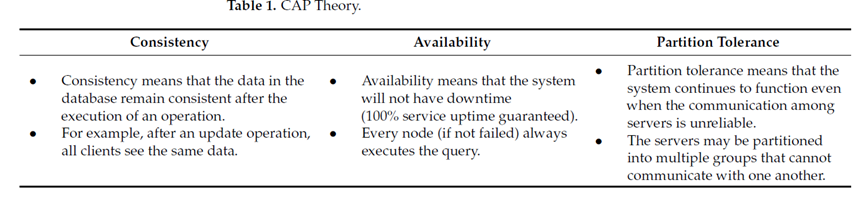
הבדל מהותי אחר בין שני בסיסי הנתונים הינו שוני בהרחבה אנכית. משמעותה של ההרחבה האנכית הינה הרחבה על ידי הוספת יותר כוח לשרת, בעוד שהרחבה אופקית משמעה הוספת שרתים נוספים. מסדי נתונים NoSQL מעוצבים באופן המקל על הרחבה אופקית על ידי פיזור הנתונים בין מספר שרתים. תחזוקת הנתונים במספר שרתים עם מסד נתונים NoSQL היא יחסית קלה משום שאין סכמת נתונים שצריך לשמור מעודכנת. הוספת יותר שרתים למסד נתונים יחסי היא מסובכת, משום שקשה לשמור על הסכימה ולשמור על עקביות הנתונים בין השרתים השונים.

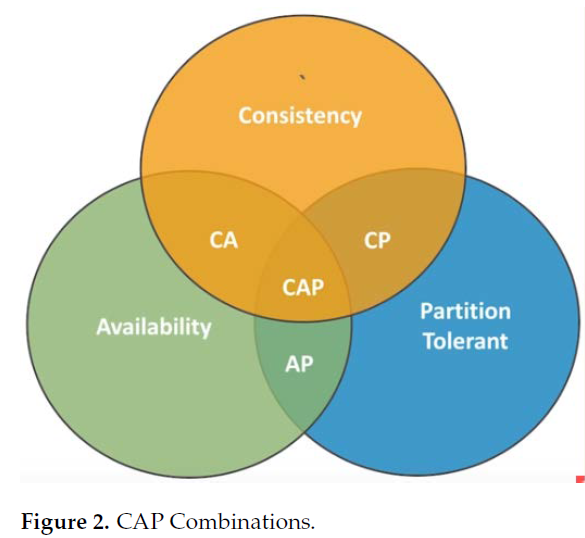
פיזור הנתונים בין מספר שרתים עוזר להימנע מנקודת כשל אחת. כשל של שרת שהוא חלק ממערכת מפוזרת לא יגרום בדרך כלל לכישלון כולל של היישום. אם מסד נתונים נשמר בשרת אחד, כישלון של שרת זה משמעותו כישלון של המערכת כולה.

ספקי מסדי נתונים מוסיפים תכונות כדי להישאר תחרותיים. מסדי נתונים SQL יכולים לאמץ תפקודים מיוחדים של NoSQL ולהפך. מצד ה-SQL ישנו מוצר בשם Postgres שמוסיף תמיכה חזקה לאחסון ולעבודה עם סוגי JSON, בעוד SQL Server עושה את אותו הדבר עבור XML. במקביל מסד נונים NoSQL בשם MongoDB הוסיפה את האופרטור lookup הדומה ל-JOIN ו-OrientDB טוענת שהיא תואמת ACID. Bozic, R, 2022).)

## תיאוריית CAP:

מאמרו של (2023) Khan, W סקר את תיאוריית CAP והתייחס לשילובה ב- NoSQL.



במחקר נבחנו שילובים מורכבים של CAP בשלושת התחומים שמצוינים בטבלה שלעיל.

תמונה 2. שילובי CAP

• מהירות: נתונים בתנועה—נתוני סטרימינג, מילי-שניות עד שניות לתגובה.

• גיוון: נתונים בצורות רבות—מובנים, לא מובנים, טקסט וכו'.

• אמיתות: נתונים מוטלים בספק—חוסר ודאות עקב השהיה, הטעיה, עמימויות וכו'.

• לא מבוסס על טבלאות ואינו משתמש ב-SQL לניהול נתונים.

• הסכימה מורכבת ממפתח-ערך, מסמך, עמודות, גרף וכו'.

• חלופה למסדי נתונים יחסיים מסורתיים.

• מסד נתונים שמטפל בנתונים לא מובנים, מבולגנים ובלתי צפויים.

• מועיל לעבודה עם מערכי נתונים מבוזרים גדולים.

המשפט של CAP אומר שמערכת מבוזרת יכולה לספק רק שניים מתוך שלוש תכונות רצויות: עקביות, זמינות וסבילות לפיצולים. Bozic, R, 2022).)

מערכת מבוזרת היא רשת ששומרת נתונים על יותר ממוקד אחד בו-זמנית. מכיוון שכל האפליקציות בענן הן מערכות מבוזרות ה CAP מאפשר ניהול נתונים יעיל ומהיר.

המשפט של CAP נקרא גם משפט ברואר, כי הוא הוצג לראשונה על ידי פרופסור אריק א. ברואר בשנת 2000.

ב- CAP ניתן לזהות עקביות שמשמעותה שכל הלקוחות רואים את אותם נתונים באותו הזמן, לא משנה לאיזה node הם מתחברים. כדי שזה יקרה, כל פעם שכתובים נתונים ל- node אחד, יש להעביר או לשכפל אותם מיד לכל שאר הnodes במערכת.

זמינות משמעותה שכל לקוח שמבצע בקשה לנתונים מקבל תגובה, אפילו אם אחד או יותר מהמוקדים לא פועלים. דרך נוספת לנסח זאת היא כך: כל ה- nodes הפועלים במערכת המבוזרת מחזירים תגובה חוקית לכל בקשה, ללא יוצא מן הכלל.

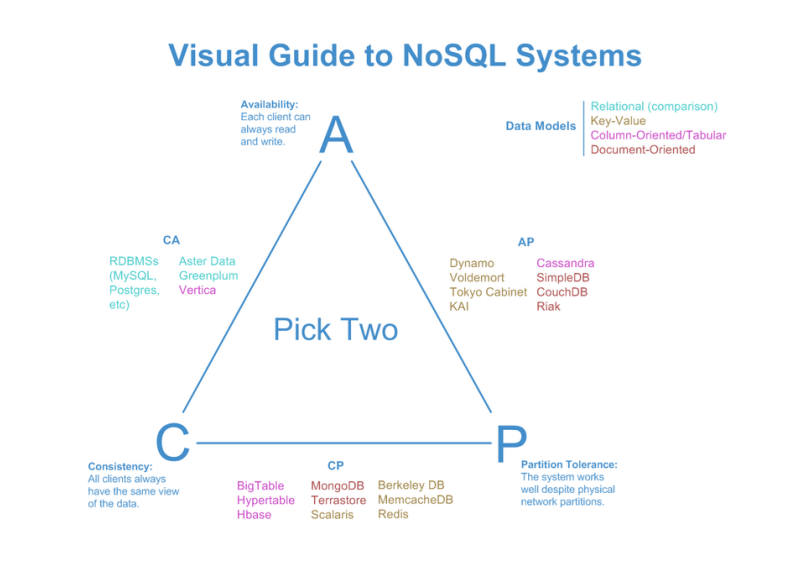
פיצול הוא קריסה בתקשורת בתוך מערכת מבוזרת, חיבור אבוד או מעוכב זמנית בין שני מוקדי. סבילות לפיצולים משמעותה שהקלסטר חייב להמשיך לפעול למרות מספר של תקלות תקשורת בין ה-nodes במערכת. Bozic, R, 2022).)

מסדי נתונים NoSQL מתאימים מאוד עבור אפליקציות ברשת מבוזרת. בניגוד למסדי נתונים SQL המאפשרים הרחבה אנכית, מסדי נתונים NoSQL מעוצבים להרחבה אופקית ומבוזרת הם יכולים להתרחב במהירות על פני רשת הולכת וגדלה המורכבת מכמה nodes מחוברים.

MongoDB היא מערכת ניהול מסדי נתונים NoSQL פופולרית ששומרת נתונים בפורמט JSON. המערכת משמשת לעיתים קרובות עבור נתונים גדולים ואפליקציות בזמן אמת הפועלות במיקומים שונים. ביחס ל- CAP, MongoDB הוא מאגר CP-הפותר פיצולים ברשת על ידי שמירה על עקביות, תוך פשרה על זמינות.

MongoDB היא מערכת עם מאסטר אחד לכל קבוצת שחזור יכול להיות רק node ראשי אחד שמקבל את כל פעולות הכתיבה. כל שאר ה nodes -באותה קבוצת שחזור הם nodes משניים שמשכפלים את יומן הפעולות של ה node -הראשי ומיישמים אותו על מערך הנתונים שלהם. כברירת מחדל, לקוחות גם קוראים מה node -הראשי, אך הם יכולים גם לציין העדפת קריאה שמאפשרת להם לקרוא מ -nodes משניים. Tiwari, S. 2011))

כאשר ה node -הראשי אינו זמין, ה node -המשני עם יומן הפעולות העדכני ביותר ייבחר כ-node הראשי החדש. לאחר שכל שאר ה nodes -המשניים משיגים את ה master -החדש, הקלאסטר הופך שוב לזמין. מכיוון שללקוחות אין אפשרות לבצע בקשות כתיבה במהלך פרק הזמן הזה, הנתונים נשארים עקביים בכל הרשת.

לסיכום,

Hurst, N. (n.d.). Visual guide to NoSQL systems. Nathan Hurst's Blog))

# החלפת נתונים עבר מסדי נתונים

## XML:

XML - Extensible Markup Language היא שפה פשוטה אך גמישה המאפשרת לתאר נתונים בצורה מובנית וניתנת לקריאה על ידי מחשבים. המבנה הבסיסי של מסמך XML הוא היררכי, בדומה לעץ הפוך, כאשר כל אלמנט יכול להכיל אלמנטים אחרים. XML נוצרה על מנת להקל על ניהול ותעבורת נתונים ברשת ובמערכות מחשב שונות, והשפעתה ניכרת על מגוון תחומים טכנולוגיים. Banzal, S., 2017))

XML אפשרה יצירה ותחזוקה של מבנים נתונים עם גמישות ויכולת התאמה, ותרמה רבות להתפתחות טכנולוגיות אינטרנטיות ומערכות מידע. בעשור האחרון, עם התקדמות הטכנולוגיה וכניסתו של JSON לשימוש נרחב, XML עשויה להיתפס פחות נפוצה, אך היא עדיין מהווה חלק חשוב ובלתי נפרד מהעולם הטכנולוגי.

XML הומצא במטרה לספק דרך סטנדרטית ועוצמתית לתאר כל סוג של נתונים. XML מציעה דרך סטנדרטית מאומצת באופן נרחב לייצוג טקסט ונתונים בפורמט שניתן לעבד ללא הרבה אינטליגנציה אנושית או מכונה. מידע בפורמט XML יכול להיות מוחלף בין פלטפורמות, שפות ואפליקציות, וניתן להשתמש בו עם מגוון רחב של כלים מועילים לפיתוח.

XML היא מטא-שפה, כלומר, היא שפה שבה נוצרות שפות אחרות. ב-XML נתונים "מסומנים" באמצעות תגים, דומה לתגים של HTML למעשה, הגרסה האחרונה של HTML, הנקראת XHTML היא שפה מבוססת XML, מה שאומר ש-XHTML עוקבת אחר כללי הסינטקס של XML . XML משמש לאחסון נתונים או מידע. נתונים אלה עשויים להיות מיועדים לקריאה על ידי אנשים או על ידי מכונות. אלה יכולים להיות נתונים בעלי מבנה גבוה כגון נתונים המאוחסנים בדרך כלל במסדי נתונים או בטבלאות אלקטרוניות, או נתונים בעלי מבנה רופף, כגון נתונים המאוחסנים במכתבים או במדריכים.

XMLעוסק בשמירה על מידע באמצעות מבנה ייחודי של סט תגים קבוע. Banzal, S., 2017)) XML נוצרה כהתפתחות טבעית משפת הסימון Standard Generalized Markup Language שהייתה קיימת בשנות ה-80 ושימשה לצורך תיעוד מסמכים והדפסה. SGML הייתה מורכבת למדי ולא תמיד נוחה לשימוש בעידן המודרני של האינטרנט. Silberschatz et al. 2010).)

XML פותחה במטרה לפשט את השימוש ב-SGML ולהפוך אותה לנגישה יותר לצרכי האינטרנט. העבודה על XML החלה בשנת 1996 בידי קבוצת העבודה של World Wide Web Consortium, שהייתה אחראית לפיתוח והטמעה של תקני אינטרנט. XML הושקה רשמית בשנת 1998 כתקן של W3C. זה היה רגע משמעותי שהביא לכניסתה של XML לשימוש נרחב ולחיזוק הסטנדרטים עבור העברת נתונים ברשת.

התקן כלל הגדרה של הסינטקס הבסיסי של XML וסטנדרטים להקפדה על מבנה נכון של מסמכים. בשנת 1999 יצא תקן XML Schema, שהוא חלק חשוב מהאקוסיסטם של XML. XML Schema מספקת דרך להגדיר את מבנה ותוכן המסמכים ב-XML בצורה פורמלית ומדויקת, ומאפשרת אימות נתונים והתאמה אישית של מבנים לפי הצורך.

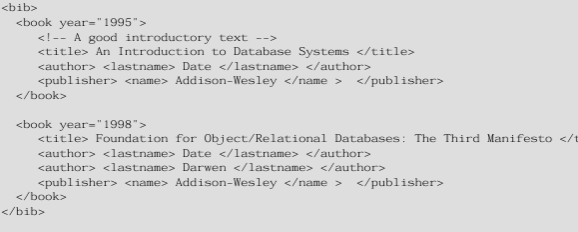
בשנת 2006 התקן Extensible Stylesheet Language Transformations שוחרר עם עדכון, אשר אפשר עיבוד והתמרה של מסמכי XML לפורמטים אחרים, כגון HTML, או XML בפורמט שונה. XSLT מאפשר המרה אוטומטית ומתקדמת של נתונים. XML הפכה לשפה נפוצה בתחום העברת נתונים בין מערכות שונות, בין אם באינטרנט ובין אם ביישומים עסקיים, ממשלתיים, אקדמיים ועוד Silberschatz et al. 2010).)

XML שימשה כבסיס לתקנים רבים, כולל RSS - Really Simple Syndication, SOAP- Simple Object Access Protocol ו-Atom. XML הופיע כסטנדרט לייצוג והחלפת מידע ב WEB ובInternet-. כתוצאה מכך, כמות גדולה של מידע מיוצגת ב-XML, ופותחו מספר כלים כדי לספק, לאחסן, לשלב ולתשאל נתוני XML.

סכימת XML המבוססת על XML Schema Definition (XSD) היא תצוגה לוגית המציגה את הקשרים בין רכיבי מטא נתונים, ובכך מגדירה את המבנה והתוכן של הנתונים על ידי יצירת כללים ותקנים עבור התוכן, כגון סוגי נתונים ואילוצים, כולל התחביר והערכים המותרים עבור אלמנטים. המטרה העיקרית של סכימת XML היא להבטיח שהנתונים תקפים ומתאימים לפורמט מסוים (Bikaki, 2024).

XML שימשה בסיס לתקנים רבים נוספים כמו XHTML (HTML המורחב באמצעות XML) ולפיתוח טכנולוגיות נוספות שנועדות לנהל ולנתח נתונים בצורה חכמה, כגון XSL (eXtensible Stylesheet Language). בשנות ה-2000 המאוחרות, JSON (JavaScript Object Notation) צבר פופולריות רבה ככלי נוסף להעברת נתונים, במיוחד באינטרנט ובפיתוח יישומים מבוססי JavaScript. JSON מספק פורמט פשוט יותר וקל יותר לקריאה ולכתיבה, אך XML נותרה חשובה ונתמכת במקרים רבים.

הרכיבים הבסיסיים של מסמך XML מורכבים מ – Tag כאשר כל אלמנט ב XML מוגדר על ידי זוג תגים. התג מגדיר את סוג הנתונים הכלולים בתוכו. ה-Content בין תג הפתיחה לסגירה נמצא התוכן של האלמנט. זה יכול להיות טקסט, מספר או אפילו אלמנטים אחרים. Attributes הן זוגות של שם וערך המופיעים בתג הפתיחה. הם מספקים מידע נוסף על האלמנט כאשר Root Element בכל מסמך XML חייב להיות בעל אלמנט שורש יחיד, אשר מכיל את כל האלמנטים האחרים. (Bikaki, 2024).

לדוגמה מבנה של XML Deutsch et al. 1998)) המכיל תגיות נתונים,

## JSON:

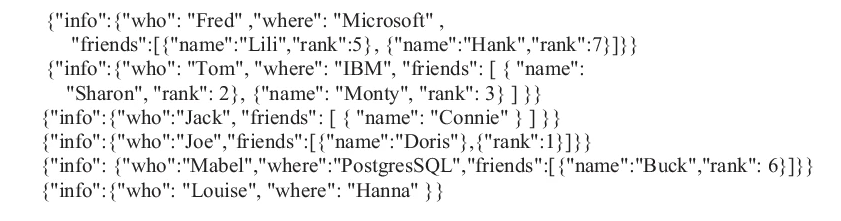
JSON (JavaScript Object Notation) הוא פורמט נתונים גמיש, פשוט ונגיש להעברת נתונים בין אפליקציות ומערכות שונות. הוא מבוסס על מבנה של אובייקטים ומערכים, ומספק דרך ברורה ומובנת לייצג נתונים בצורה מבנית וארגונית. JSON הפך לתקן נפוץ בתקשורת בין שרתים ללקוחות באינטרנט ובפיתוח יישומים שונים. JSON נוצר בשנת 1999 על ידי Douglas Crawford, אשר הציע אותו כפורמט פשוט ונגיש לייצוג אובייקטים ולתיאום נתונים בין מערכות. Crawford פיתח את JSON כחלופה לפורמטים הקיימים והכבדים יותר כמו XML.

JSON Schema הוא תקן מתפתח לתיאור משפחות של מסמכי JSON. זוהי שפה לוגית, המבוססת על קבוצה של קביעות המתארות תכונות של ערך ה-JSON בניתוח ועל קומבינטורים לוגיים או מבניים עבור קביעות אלו, כולל אופרטור שלילה (Baazizi, 2023).

JSON הוכר באופן רשמי על ידיInternet Engineering Task Force תחת שם RFC 4627. RFC זה קובע את התקנים הבסיסיים של JSON ומספק הנחיות כיצד להשתמש בו. בשנת 2013 התקן JSON קיבל עדכון נוסף עם הוצאת RFC 7159 עדכון זה הביא לתקן הנוכחי שהוסיף שיפורים כמו טיפול נכון במחרוזות ותחביר מסודר יותר. בשנת 2017 RFC 8259 פורסם, ועדכון זה שמר על ההתאמה והבהירות של התקן, כולל עדכונים קלים בחוקיות ובתחביר.

JSON הוא פורמט טקסטואלי קל משקל המשמש להחלפת מידע בין יישומים שונים. הוא מבוסס על תת-קבוצה של שפת JavaScript, אך הוא עצמאי משפה ונתמך על ידי רוב שפות התכנות המודרניות. JSON משמש בעיקר להעברת נתונים בין שרתים ללקוחות, כמו גם לאחסון הגדרות של יישומים. פורמט JSON מאוד פופולרי בקרב מפתחים הודות למבנה פשוט וקל להבנה עבור בני אדם, מה שמקל על כתיבת וניתוח של נתונים בפורמט זה. (Baazizi, 2023).

JSON הוא פורמט קומפקטי ויעיל, מה שמפחית את כמות הנתונים המועברים ברשת. רוב שפות התכנות המודרניות כוללות ספריות או מודולים מובנים לתמיכה ב-JSON, מה שמקל על השימוש בו. מבנה בסיסי של JSON מבוסס על שני מבנים עיקריים כגון אובייקטים ומערכים.

לדוגמה , איור המציג קבוצת מסמכי JSON בשם "info", המתארים אנשים. בשמות האנשים, בשייכותם ובחבריהם מתוארים מאפיינים. השמות המתאימים ("מי", "חברים" ו"איפה") מכונים מאפיינים. לדוגמה, אדם אחד, פרד, מועסק במיקרוסופט ויש לו חברים בשם לילי וחנק. Petković, D. 2017))

# כלים לביצוע פעולות וחישובים על הנתונים

## Java Stream:

ביג דאטה הוא מונח המשמש לתיאור אוסף של יישומים גדולים ומורכבים שקשה לעבד אותם באמצעות טכניקות עיבוד נתונים מסורתיות, או בגלל היקף הנתונים העצום שמיוצר, או בגלל דרישות זמן קפדניות המוטלות על עיבוד הנתונים. לדוגמה, מאיץ החלקיקים הגדול (Large Hadron Collider) יכול להפיק זרם נתונים גולמי בהיקף של כ-1 פטה-בייט לשנייה (X. C. Vidal & R. C. Manzano, 2014). נתונים אלו חייבים לעבור סינון לפני אחסון, מה שמציב דרישה לזמן הסינון ושלב האחסון. בתחומים מסוימים, בקשות לניתוח וכריית נתונים מאוחסנים חייבות להתבצע מספיק מהר עבור המשתמש הסופי. במצבים כאלו, עיבוד מהיר יותר מאפשר דיוק פוטנציאלי גדול יותר.( Chan et al. (2017).

כדי להתמודד עם האתגרים הללו, מערכות מחשוב מורכבות זוכות לעניין גובר בקהילת המחקר ובתעשייה, ומשמשות במגוון רחב של יישומים כמו עיבוד נתונים גדולים, ארכיטקטורות תוכנה, הנדסת תוכנה ומערכות קריטיות לבטיחות. במערכות אלה, הביצועים חשובים מאוד, ולכן יש לפתח קוד שמנצל היטב את הארכיטקטורות המודרניות עם כמה ליבות. זהו אתגר שקשור ישירות לדרישות הביצועים של יישומים בביג דאטה, שכן קוד יעיל יכול לשפר את המהירות והדיוק של עיבוד הנתונים. יחד עם זאת, קיימים אתגרים רבים במקביל את הקוד, כדי לשמור על תחזוקה נוחה וסקלביליות, מה שמקשה על המפתחים. לכן, מספר שפות תכנות (כמו Java (Stream מציעות מודלים פשוטים ותיאורתיים, המאפשרים להקל על המפתחים בהפיכת חישובים סיקוונציאליים לחישובים מקביליים בקלות (Basso et al., 2022).

Stream הוא כלי/ספרייה שהוצגה ב-Java 8 המאפשר לנו להתייחס לאוספים כאל רצפים ומאפשר לנו לבצע פעולות וחישובים על האלמנטים הנמצאים באוסף (מערך, רשימה, מפה) .ה-Stream API נוספה ב-Java 8 כדי לאפשר למפתחים לבצע עיבוד נתונים בסגנון דקלרטיבי ותמציתי, המבוסס על תכנות פונקציונלי. כלי זה מציע שתי אבחנות עיקריות: הראשונה, הזרם (stream), שמייצגת רצף של אלמנטים שמקורם במקורות נתונים שונים; והשנייה, צנרת הזרם (pipeline), שהיא רצף של פעולות המיועדות ליישום על האלמנטים בזרם עם הביצוע. פעולות אלו כוללות מיפויים (mapping), חיפושים (searching), סינון (filtering) והפחתת נתונים (reduction) (Basso et al., 2022).

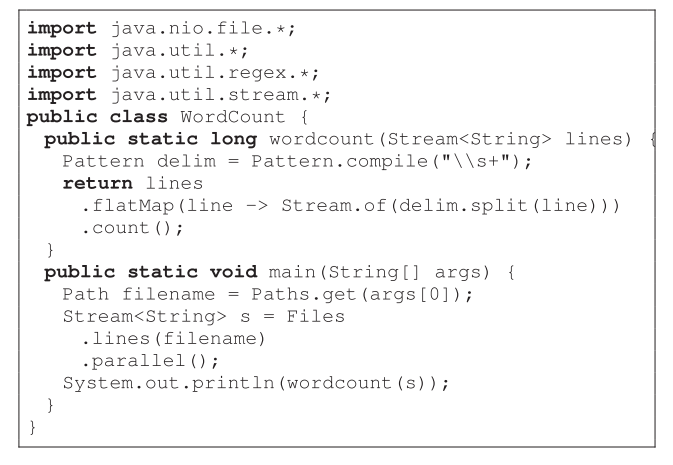
היתרונות של ה-Stream API כוללים פשטות וקריאות, המאפשרים כתיבה של קוד קריא ופשוט יותר. זה מקל על המפתחים להגדיר את ההתנהגות הרצויה של הקוד מבלי להיכנס לפרטים של ניהול לולאות (loops). יתרון מרכזי נוסף הוא תמיכה בעיבוד מקבילי (parallel processing), המאפשר להפעיל פייפליינים של פעולות בצורה מקבילה בקלות, מה שמנצל את היתרונות של מעבדים מרובי ליבות (multi-core processors). כך, ניתן להשיג שיפוט משמעותי בעיצוב התוכנה, להקל על הרחבת והחזקת היישום, ולייעל ביצועים (Rosales et al., 2022).

בנוסף, ה-Stream API מאפשרת אופטימיזציה של ביצועים באמצעות טכניקות כמו אינליינינג (inlining) והמרת פייפליינים ללולאות, דבר שמפחית את העומס הנוסף שנגרם משימוש בפונקציות אנונימיות. שימושים נפוצים של ה-Stream API כוללים עיבוד נתונים ממקורות שונים כמו אוספים, מערכים ופונקציות גנרטור (generator functions), וכן מניפולציה של נתונים, כגון מיפוי, סינון וצברת נתונים בצורה פשוטה ויעילה. יישומים המתאימים לכך הם בעיקר בתחום עיבוד נתונים גדולים (big data) או יישומים עם דרישות ביצועים גבוהות (Basso et al., 2022).

עם זאת, ישנם חסרונות בשימוש ב-Stream. השימוש בה עלול לגרום לעומס ריצה משמעותי, במיוחד כאשר מדובר בזרמים מקבילים, דבר שעלול להוביל לביצועים פחות טובים בהשוואה לקוד אימפרטיבי. קושי נוסף הוא באופטימיזציה; למרות שה-Stream מציעה אופטימיזציה, ישנם מקרים שבהם לא ניתן לנצל את כל היתרונות, במיוחד עם זרמים מקבילים. לא כל הטכניקות והכלים לאופטימיזציה של קוד אימפרטיבי מתאימים לזרמים, מה שמוביל לפספוס הזדמנויות אופטימיזציה. (Basso et al., 2022)

לסיכום, ה-Stream API ב-Java מציעה יתרונות משמעותיים בשיפור קריאות הקוד ובפשטות העיבוד, אך יש לקחת בחשבון את החסרונות הפוטנציאליים בביצועים ובאופטימיזציה, במיוחד כאשר עובדים עם זרמים מקבילים.

לדוגמה, יישום פשוט של ספירת מילים באמצעות Java Stream 8.( Chan et al. (2017).



## Apache Spark:

ניתוח נתוני ביג דאטה הוא אחד מתחומי המחקר הפעילים ביותר, עם הרבה אתגרים וצרכים לחדשנות שיכולים להשפיע על מגוון רחב של תעשיות. כדי לממש את הדרישות החישוביות של ניתוח נתונים מאסיבי, חשוב ליצור מסגרת יעילה לתכנון, יישום וניהול של הצינורות והאלגוריתמים הנדרשים. בהקשר זה, Apache Spark צמחה כמנוע מאוחד לניתוח נתונים בקנה מידה גדול עבור מגוון רחב של עומסים. היא הציגה גישה חדשה למדע נתונים והנדסה שבה ניתן לפתור מגוון בעיות נתונים באמצעות מנוע עיבוד אחד בשפות כלליות. בעקבות מודל התכנות המתקדם שלה, Apache Spark אומצה כמסגרת מהירה וסקלאבילית גם באקדמיה וגם בתעשייה. היא הפכה לפרויקט הקוד הפתוח הפעיל ביותר בתחום הביג דאטה ואחד הפרויקטים הפעילים ביותר ב- Apache Software Foundation. Salloum et al. 2016)).

ספארק הושק לראשונה על ידי מתאי זאהריה ב-AMPlab של אוניברסיטת ברקלי בשנת 2009 ונפתח כקוד פתוח בשנת 2010 תחת רישיון BSD. לאחר מכן, הפרויקט הועבר לקרן תוכנת אפאצ'י בשנת 2013. מספר פרויקטי מחקר תרמו תרומות משמעותיות לבניית ושיפור הליבה של ספארק וספריות העל הראשיות. לדוגמה, פיתוח MLlib של ספארק החל מפרויקט MLbase, ולאחר מכן פרויקטים נוספים החלו לתרום (כגון KeystoneML). Spark SQL התחיל מפרויקט Shark, והפך לספרייה חיונית ב-Apache Spark. גם GraphX התחיל כפרויקט מחקר ב-AMPlab והפך לחלק מפרויקט Apache Spark מאז גרסה 0.9.0. מספר חבילות גם נתרמו ל-Apache Spark הן מהאקדמיה והן מהתעשייה. בנוסף, מייסדי Apache Spark הקימו את Databricks, חברה המעורבת באופן הדוק בפיתוח Apache Spark. (Armbrust et al. (2015), Gonzalez (2014), Meng et al. (2016), Xin et al. (2013), Bosagh Zadeh et al. (2016), Zaharia (2016), Zaharia et al. (2010), Zaharia et al. (2014), Zaharia et al. (2013).)

מערכת Apache Spark מורכבת ממספר רכיבים עיקריים כולל הליבה של Spark ורכיבי העל: MLlib של ספארק למכונת למידה, GraphX לניתוח גרפים, Spark Streaming לעיבוד זרמים ו-Spark SQL לעיבוד נתונים מובנים. המערכת מתפתחת במהירות עם שינויים ב-APIs הליבה והוספת ספריות על. האבטיפוס המרכזי של נתונים, ה-RDD (Resilient Distributed Dataset), פותח את הדלת לעיצוב אלגוריתמים וצנרות נתונים ניתנים להרחבה עם ביצועים טובים יותר. עם שיתוף הנתונים היעיל של ה-RDD ומגוון הפעולות, ניתן לתכנן ולבצע עבודות שונות בצורה יעילה. בעוד שה-RDD היה האבטיפוס המרכזי שהוצג בספארק 1.0 דרך ה-RDD API, הייצוג של קבוצות נתונים היה תחום פעיל לשיפור בשנתיים האחרונות. אלטרנטיבה חדשה, ה-DataFrame API, הוצגה בספארק 1.3, ואחריה הוצג preview של ה-Dataset API בספארק 1.6. (Zaharia (2016), Zaharia et al. (2010), Zaharia et al. (2014), Zaharia et al. (2013).)

Apache Spark התפתח כתקן הדיפולטיבי לניתוח נתונים גדולים לאחר Hadoop’s MapReduce. כמסגרת, הוא משלב מנוע ליבה לעיבוד מפוזר עם מודל תכנות מתקדם לעיבוד בזיכרון. אף על פי שיש לו את אותן יכולות סקלביליות ליניאריות וסיבולת בעיות כמו MapReduce, הוא מציע מודל תכנות בזיכרון שלב-אחר-שלב, בניגוד למודל הקשיח של מיפוי-ולאחר-מכן-הפחתה מבוסס דיסק. עם מודל כזה, Apache Spark מהיר בהרבה וקל יותר לשימוש. הוא מגיע עם APIs עשירים בכמה שפות Scala, Java, Python, SQL) ו-(R לביצוע פעולות מפוזרות מורכבות על נתונים מפוזרים. בנוסף, Apache Spark עושה שימוש בזיכרון של קלאסטר המחשוב כדי להפחית את התלות במערכת הקבצים המפוזרת, מה שמוביל לשיפוט ביצועים דרמטי בהשוואה ל-Hadoop MapReduce. הוא נחשב גם כמנוע כללי שמעבר ליישומים מבוססי-קבוצה, משלב סוגים שונים של חישובים (כגון קבוצות עבודות, אלגוריתמים חזרתיים, שאילתות אינטראקטיביות ועיבוד זרמים) שדרשו בעבר מערכות מפוזרות נפרדות. הוא מתבסס על האבטיפוס של RDDs המספק שיתוף נתונים יעיל בין חישובים. במסגרת קודמת, חסרו מסגרות זרימת נתונים יכולת שיתוף נתונים כזו, אף על פי שהיא דרישה חיונית עבור עבודות שונות. (Karau et al. 2015)

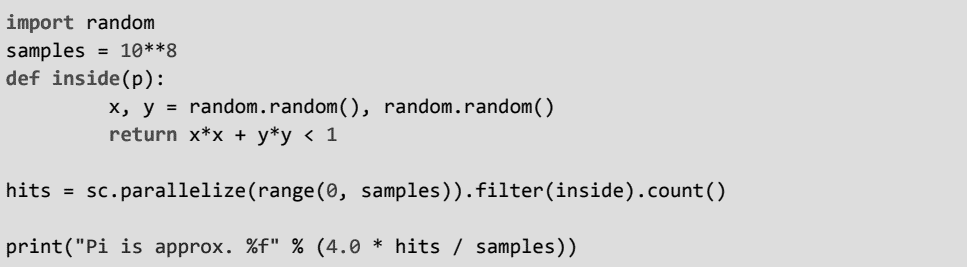
כמנוע הדור הבא לניתוח נתונים גדולים Apache Spark, יכול להקל על אתגרים מרכזיים בעיבוד נתונים, אלגוריתמים, ניתוח אינטראקטיבי וניתוח תפעולי, בין היתר. עם Apache Spark, ניתן לעבד נתונים דרך גרף מכוון לא מעגלי (DAG) של מפעילים תוך שימוש בערכות עשירות של טרנספורמציות ופעולות. הוא מפיץ אוטומטית את הנתונים ברחבי הקלאסטר ומבצע את הפעולות הנדרשות במקביל. הוא תומך במגוון טרנספורמציות שמקלה על עיבוד נתונים, במיוחד כאשר קשה יותר לבדוק קבוצות נתונים גדולות. מצד שני, קבלת תובנות מועילות מנתונים גדולים דורשת ניסויים בשלב שונים כדי לבחור את התכונות, השיטות, הפרמטרים ומדדי ההערכה הנכונים. Apache Spark מיועד באופן טבעי להתמודד עם סוג כזה של עיבוד חזרתי, שדורש יותר מעברת נתונים אחת על אותה קבוצת נתונים (למשל, MLlib לעיצוב והתאמת אלגוריתמים וצנרות למידת מכונה. Salloum et al. 2016)).

בנוסף לאלגוריתמים Apache Spark, מתאים מאוד לניתוח אינטראקטיבי, שיכול להגיב במהירות לשאילתות המשתמש על ידי סריקת קבוצות נתונים בזיכרון המפוזר. יתרה מכך, Apache Spark אינו רק מנוע מאוחד לפתרון בעיות נתונים שונות במקום ללמוד ולתחזק מספר כלים שונים, אלא גם מסגרת כללית המקצר את הדרך מניתוח הסברתי במעבדה לניתוח תפעולי ביישומי נתונים ובמסגרות. כתוצאה מכך, זה יכול להוביל לעלייה בפרודוקטיביות של האנליסטים, במיוחד כאשר ספריות העל שלו משולבות ליישום אלגוריתמים וצנרות מורכבות. Ryza et al. 2017)).

למרות זאת, החסרונות של Apache Spark כוללים כמות גדולה של פרמטרי קונפיגורציה, עם יותר מ-180 פרמטרים, מה שהופך את הבחירה בקונפיגורציה האופטימלית לתהליך מורכב ומאתגר. בנוסף, קיימות אינטראקציות מורכבות בין הפרמטרים השונים, שמקשות על ניבוי ביצועי היישומים בצורה מדויקת. הביצועים של Spark תלויים באופן משמעותי באופי היישום, ולכן שימוש בקונפיגורציה גנרית עשוי להוביל לביצועים תת-אופטימליים. כאשר מערכי הנתונים גדולים, זמן הריצה של היישום עשוי להיות ארוך מאוד, דבר המקשה על ניתוח וביצוע התאמות בזמן אמת. כמו כן, מודלים קיימים לניבוי ביצועים מתמקדים לעיתים קרובות בפרמטרים בודדים ואינם מצליחים להתמודד בצורה יעילה עם מספר רב של פרמטרים או קונפיגורציות מורכבות, מה שמוביל לדיוק מוגבל בניבוי הביצועים. Cheng et al. 2021))

לסיכום, Apache Spark הוא מנוע לניתוח נתונים בקנה מידה גדול, המציע יתרונות משמעותיים לעיבוד מהיר וסקלאבילי של נתוני ביג דאטה, אך יחד עם זאת, מציב אתגרים מורכבים בשל כמות פרמטרי הקונפיגורציה והקושי בניבוי ביצועים מדויקים בפרויקטים מורכבים.

לדוגמה חישוב PI ע"י שימוש בSpark באמצעות פייתון.



# סיכום ומסקנות

מסדי נתונים NoSQL ורלציוניים הם כלי מרכזי בניהול נתונים בעידן המודרני, וכל אחד מהם מציע יתרונות שונים שמתאימים לצרכים מגוונים בארגונים וביישומים שונים. מסדי נתונים NoSQL הפכו לפופולריים בזכות גמישותם הגבוהה והיכולת שלהם לנהל סוגי מידע מגוונים, הכוללים מידע מובנה, חצי-מובנה ולא-מובנה. מסדי נתונים אלה מצטיינים בעבודה ללא סכמה קבועה, מה שמאפשר למפתחים לשנות את מבנה הנתונים בקלות, ללא השפעה על הנתונים הקיימים. גמישות זו היא קריטית במיוחד בסביבות כמו Big Data, IoT ומערכות ניהול תוכן, שבהן יש צורך לעבד כמויות גדולות של נתונים מגוונים במהירות וביעילות. לצד היתרונות הללו, מסדי נתונים NoSQL מציעים גם ביצועים גבוהים, מה שהופך אותם לאידיאליים ליישומים בזמן אמת. חלקם קוד פתוח, דבר המאפשר לארגונים לחסוך בהוצאות ולנהל את הנתונים בקנה מידה גדול. עם זאת, קיימים אתגרים שיש לקחת בחשבון, כמו רגולציות מחמירות המצריכות איזון בין הצרכים העסקיים לפרטיות הנתונים. בבחירת מסד נתונים NoSQL, יש לקחת בחשבון את סוגי הנתונים שצריכים להיות מנוהלים ואת הדרישות השונות של הארגון.

מאידך, מסדי נתונים רלציוניים מאפשרים גישה נוחה ויעילה לנתונים באמצעות טבלאות המייצגות ישויות שונות, כמו לקוחות ומוצרים. כל טבלה מורכבת משורות ועמודות, כאשר כל שורה מייצגת רשומה ייחודית והעמודות מייצגות תכונות שונות של ישות זו. מערכת ניהול מסדי הנתונים הרלציוניים (RDBMS) מבטיחה שלמות ועקביות של הנתונים באמצעות מפתחות ראשיים, שמספקים קישורים בין טבלאות. שפת SQL, שהיא הסטנדרטית לניהול מסדי נתונים רלציוניים, מאפשרת ביצוע פעולות מגוונות כמו שאילתות, עדכונים והגדרת מבנים חדשים. אלגברה רלציונית, כבסיס התיאורטי למודלים אלו, מספקת שיטות לפעולות על נתונים ומסייעת בהבנת הממשקים המורכבים בין טבלאות .

בסופו של דבר, מסדי נתונים NoSQL ורלציוניים מציעים פתרונות מגוונים לאתגרים של ניהול נתונים, והבחירה בין השניים תלויה בצרכים העסקיים ובדרישות המידע. כל מסד נתונים מציע יתרונות וחסרונות שחשוב להבין בטרם קבלת החלטה, מה שמדגיש את החשיבות של ניתוח והבנת צרכי הארגון לפני בחירת פתרון ניהולי.

תמונה שמכילה טקסט, צילום מסך, מספר, גופן

התיאור נוצר באופן אוטומטיבין סוגי מסדי הנתונים NoSQL, ניתן למצוא קטגוריות שונות כמו מבוססי מפתח-ערך, Wide-Column Stores, מסדי נתונים מסוג מסמכים ומאגרי נתונים גרפיים, וכל אחת מהן מציעה יתרונות וחסרונות שונים.

הטבלה משווה מפתח-ערך, עמודות, מסדי נתונים מסוג מסמכים ומאגרי נתונים גרפיים בהתבסס על תכונות לא-פונקציונליות כמו: ביצועי שאילתות, מדרגיות של נתונים, גמישות סכֵמָה, מבנה מאגר הנתונים ומורכבות הערכים. הטבלה מתארת כי עבור נתונים פשוטים שניתן לייצג בקלות כזוגות מפתח-ערך, ניתן לבחור מסדי נתונים מסוג מפתח-ערך מכיוון שהיא תספק ביצועים גבוהים, מדרגיות וגמישות. אם ניתן לייצג את הערכים בצורה של עמודות והם חצי-מובנים, אז מסדני נתונים עמודות היא הבחירה המתאימה מכיוון שהיא תספק ביצועים גבוהים ומדרגיות. אם הנתונים ניתנים לייצוג בפורמט JSON, יש להעדיף במסדי נתונים מסוג מסמכים שכן היא מספקת ביצועים גבוהים, גמישות ובדרך כלל מדרגיות גבוהה. אם יש צורך לאחסן נתונים שניתן לייצגם באמצעות תורת הגרפים או אם הנתונים מקושרים מאוד זה לזה, אז נשתמש במסדי נתונים גרפים, אשר מספק יציבות גבוהה, אך הביצועים והמדרגיות משתנים(Gupta, A., et al. 2017)).

על כן, מבוססי מפתח-ערך, כמו Redis, מתמקדים באחסון נתונים בצורה של זוגות מפתח-ערך. זה מאפשר גישה מהירה ויעילה לנתונים, במיוחד עבור פעולות קריאה וכתיבה. הפשטות של מבנה הנתונים במערכות אלו תורמת לביצועים גבוהים, אך ישנם חסרונות כמו חוסר תמיכה בשאילתות מורכבות, מה שעשוי להגביל את השימוש בהן במקרים בהם יש צורך בניתוח נתונים מתקדם.

עם זאת, Wide-Column Stores, כמו Cassandra, מיועדים לעבודה עם נתונים גדולים ומאפשרים גמישות מבנית. הנתונים מאורגנים במבנה של "משפחות עמודות", שמאפשר לכל שורה להכיל מספר שונה של עמודות. היתרון המרכזי של מערכות אלו הוא הביצועים הגבוהים בכתיבה וקריאה, עם אפשרות להתרחבות אופקית. עם זאת, קיימת בעיה בעקביות הנתונים, והניהול של קלאסטרים עלול להיות מאתגר ודורש ידע טכני מעמיק.

כמו כן, מסדי נתונים מסוג מסמכים, כמו MongoDB , מתמקדים בניהול נתונים לא מובנים באמצעות פורמטים כמו JSON. הם מציעים גמישות רבה בעבודה עם נתונים דינמיים ומורכבים, עם ביצועים מהירים ותמיכה בהתרחבות. עם זאת, קיימים אתגרים בתחום האבטחה ושיתוף הפעולה, מה שעלול להשפיע על השימוש במערכות אלו בארגונים רגישים.

בהשוואה לכך, מאגרי נתונים גרפיים, כמו Neo4j , מיועדים לניתוח קשרים בין נתונים בצורה של גרפים, כולל צמתים, קשרים ותכונות. הם מתאימים במיוחד למערכות שבהן הקשרים בין הנתונים הם מורכבים, כמו ברשתות חברתיות. היתרון המרכזי של מערכות אלו הוא הביצועים הגבוהים בשאילתות המערבות קשרים רבים, מה שמקנה להן ערך מוסף ביישומים הדורשים ניתוח קשרים.

לכן, כשמדברים על מערכות כמו MongoDB, Cassandra, Redis ו-Neo4j, יש להבין שהבחירה במערכת המתאימה תלויה בצרכים הספציפיים של היישום ובדרישות הביצועים והגמישות. MongoDB מציעה גמישות רבה בעבודה עם נתונים לא מובנים, אך חוסר הסכמה קבועה עלול להוביל לבעיות בהתמדה של נתונים. Cassandra מיועדת לארגונים שצריכים לעמוד בדרישות זמינות קפדניות, ומציעה ביצועים מצוינים בניהול כמויות גדולות של נתונים, אך דורשת ניהול טכני מורכב. Redis מציעה מהירות גבוהה במיוחד, אך עשויה להיות יקרה כשמדובר בכמויות גדולות של נתונים. Neo4j, לעומת זאת, מציעה פתרון מצוין לניתוח קשרים ומספקת שפת שאילתות אינטואיטיבית המקלות על העבודה עם נתונים גרפיים.

לפיכך, מסדי נתונים NoSQL מציעים פתרונות חדשניים וגמישים לאתגרים של ניהול נתונים מודרניים. הבחירה במערכת הנכונה תלויה בהבנה מעמיקה של המאפיינים והצרכים הספציפיים של הארגון, ועל מנת להבטיח פתרון טכנולוגי שיתמוך באתגרים ובדרישות המורכבות של עולם המידע המודרני.

נוסף על כך, Elasticsearch הוא מנוע חיפוש וניתוח נתונים מתקדם, המיועד להתמודד עם סוגים שונים של נתונים, כולל נתונים מובנים ולא מובנים. הוא מציע יכולות חיפוש בזמן אמת, ממשק API מבוסס REST, וביצועים גבוהים עם זמני תגובה נמוכים. יתרונותיו כוללים סקלאביליות גבוהה, אינדקס הפוך לחיפושים מהירים, ותמיכה בנתונים גיאוגרפיים. עם זאת, ישנם חסרונות כמו מורכבות בהגדרה, צורך בתחזוקה מתמדת ואתגר בכתיבת שאילתות מורכבות. בנוסף, השימושים הנפוצים ב-Elasticsearch כוללים שיפור חוויית החיפוש באתרים וניתוח נתונים ממקורות שונים. במקביל, RDF (Resource Description Framework) הוא מודל נתונים שמטרתו לייצג מידע על משאבים באינטרנט בצורה גמישה ומבנית. הוא מבוסס על טריפלטים, המאפשרים ייצוג קשרים מורכבים בין משאבים. יתרונותיו כוללים גמישות בייצוג נתונים, קלות בשיתוף מידע בין מערכות שונות ותמיכה בסמנטיקה. עם זאת, יישום RDF עשוי להיות מורכב ודורש הבנה מעמיקה, וביצועים של שאילתות SPARQL עשויים להיחלש עם כמויות גדולות של נתונים.

הודות לכך, הבחירה בין שתי הטכנולוגיות, תלויה בצרכים הספציפיים של הארגון. Elasticsearch מציע ביצועים גבוהים וניהול יעיל של חיפושים, בעוד RDF מציע גמישות וסמנטיקה בייצוג נתונים. שני הפתרונות דורשים הבנה טכנית, והכשרה עשויה להיות נחוצה כדי לנצל את מלוא הפוטנציאל של כל טכנולוגיה. ניתן למצוא יישומים שונים לכל טכנולוגיה; Elasticsearch מתמקד בשיפור חוויית החיפוש וניתוח נתונים בזמן אמת, בעוד RDF מיועד ליצירת רשתות מידע מחוברות. יש לקחת בחשבון את האתגרים הטכניים הכרוכים בשימוש בכל פתרון, במיוחד עבור ארגונים קטנים או כאלה עם משאבים מוגבלים.

למעלה מזה, XML (Extensible Markup Language) היא שפה מבנית המיועדת לתיאור נתונים באופן גמיש ומסודר. המבנה ההיררכי שלה מאפשר אחסון מידע בצורה נוחה, והיכולת של XML להחליף נתונים בין פלטפורמות שונות הפכה אותה לסטנדרט בתחום העברת הנתונים. XML נוצרה במטרה לפשט את השימוש ב-SGML, והושקה רשמית בשנת 1998. מאז, XML שימשה כבסיס לתקנים רבים כמו RSS ו-SOAP, והיא עדיין חשובה לעבודה עם נתונים מורכבים.

אי לכך, למרות שהופיעה תחרות מצד JSON, XML שומרת על מקומה בזכות יכולות האימות והגמישות שלה. JSON (JavaScript Object Notation) הוא פורמט נתונים קל ומשתמש המיועד להעברת מידע בין מערכות שונות. JSON מציע מבנה ברור, מבוסס על אובייקטים ומערכים, מה שהופך אותו לפופולרי במיוחד בפיתוח יישומים אינטרנטיים. JSON פותחה בשנת 1999 כהחלפה לפורמטים כבדים כמו XML, וזכתה להכרה רחבה עם תקנים רשמיים שהובילו לשיפורים ושמירה על תחביר מסודר.

יתר על כן, JSON קל להבנה ונוח למפתחים, ולכן הוא הפך לפורמט מועדף להעברת נתונים בין שרתים ללקוחות. הבחירה בין XML ל-JSON תלויה בצרכים הספציפיים של הפרויקט. XML מציעה יתרונות במבנים מורכבים ודורשת תקני אימות מחמירים, בעוד JSON מציעה מבנה פשוט ונגיש, במיוחד עבור יישומים מודרניים. XML מציעה גמישות גבוהה באחסון נתונים, אך עשויה להיות מסורבלת לעיתים. JSON, עם זאת, מספק פתרון קל ומהיר, מה שהופך אותו למועדף בפיתוחים מודרניים, במיוחד באינטרנט.

לבסוף, שתי השפות נתמכות על ידי רוב שפות התכנות המודרניות, אך JSON נהנית מתמיכה רחבה יותר בפיתוח יישומים מבוססי JavaScript. בסופו של דבר, הבחירה בין XML ל-JSON צריכה להתבצע על פי דרישות הסביבה, סוג הנתונים וצרכי ההעברה, תוך שמירה על גמישות ויכולת התמודדות עם שינויים עתידיים.

בנוגע לעיבוד נתוני ביג דאטה, Java Stream, שהוצג ב-Java 8, מציע דרך נוחה ופונקציונלית לעיבוד אוספי נתונים באמצעות פעולות כמו סינון, מיפוי והפחתה. היתרון המרכזי הוא הפשטות והקריאות של הקוד, המאפשרות עיבוד מקבילי בקלות, דבר המנצל את יכולות המעבדים מרובי הליבות. עם זאת, ישנם חסרונות כמו עומס ריצה אפשרי ואופטימיזציה לא תמיד יעילה, במיוחד בעבודה עם זרמים מקבילים.

מנגד, Apache Spark נחשב למנוע עיבוד נתונים מתקדם וסקלאבילי, המיועד לניתוח נתוני ביג דאטה. הוא מציע ביצועים גבוהים ויכולת עבודה עם אלגוריתמים מורכבים במגוון שפות. Spark מספק פתרון כולל, מפשט את העבודה עם נתונים ומקצר את הזמן הנדרש לניתוח תפעולי. יחד עם זאת, יש לו חסרונות, כמו מספר רב של פרמטרי קונפיגורציה שעשויים להקשות על ההתאמה האופטימלית וביצועים תת-אופטימליים במקרים מסוימים.

לפיכך, שתי הטכנולוגיות מציעות יתרונות משמעותיים בעיבוד נתוני ביג דאטה, אך יש לשקול את האתגרים שהן מציבות, כמו מורכבות הקונפיגורציה בבחירת Spark והקשיים באופטימיזציה של Java Stream. המפתח להצלחה בעיבוד נתוני ביג דאטה הוא לבחור את הכלים המתאימים ולהבין את המגבלות שלהם בהתאם לדרישות הספציפיות של הפרויקטים.

לסיכום, ניהול נתונים הוא תחום מורכב ודינמי, והבחירה במערכת המתאימה תלויה בצרכים העסקיים הספציפיים ובדרישות השונות של הארגון. ארגונים צריכים לבחור במערכת הניהול המתאימה ביותר לצרכים שלהם, תוך כדי הבנת היתרונות והחסרונות של כל סוג. חשוב לקחת בחשבון את סוגי הנתונים שברצונם לנהל, דרישות הביצועים והגמישות. כדי לנצל את מלוא הפוטנציאל של מסדי הנתונים NoSQL ורלציוניים, יש צורך בהכשרה והבנה מעמיקה של הטכנולוגיות. השילוב בין פתרונות טכנולוגיים שונים, כמו Elasticsearch ו-RDF, יכול לסייע בשיפור ניהול הנתונים. הארגונים צריכים להיות מודעים לאתגרים הטכניים שיכולים לצוץ בשימוש במערכות אלו, במיוחד כשמדובר בכמויות גדולות של נתונים או בדרישות אבטחה ורגולציה מחמירות. חשוב להבין את היתרונות של הגמישות שמסדי נתונים NoSQL מציעים, במיוחד בסביבות משתנות, מול היתרונות של המבנה הקבוע של מסדי נתונים רלציוניים, כדי לקבוע את המערכת האופטימלית לניהול נתונים.

# ביבליוגרפיה

Agrawal, S., & Patel, A. (2016). Astudy on graph storage database of nosql. \*International Journal on Soft Computing, Artificial Intelligence and Applications (IJSCAI)\*, 5(1), 33-39.

Almeida, D., Lopes, M., Saraiva, L., Abbasi, M., Martins, P., Silva, J., & Váz, P. (2023, August). Performance Comparison of Redis, Memcached, MySQL, and PostgreSQL: A Study on Key-Value and Relational Databases. In \*2023 Second International Conference On Smart Technologies For Smart Nation (SmartTechCon)\* (pp. 902-907). IEEE.

Angles, R., Thakkar, H., & Tomaszuk, D. (2020). Mapping RDF databases to property graph databases. \*IEEE Access\*, 8, 86091-86110.

Anusha, K., Rajesh, N., Kavitha, M., & Ravinder, N. (2021, April). Comparative Study of MongoDB vs Cassandra in big data analytics. In \*2021 5th International Conference on Computing Methodologies and Communication (ICCMC)\* (pp. 1831-1835). IEEE.

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., ... & Zaharia, M. (2015, May). Spark sql: Relational data processing in spark. In Proceedings of the 2015 ACM SIGMOD international conference on management of data (pp. 1383-1394).

Avi Silberschatz, Henry F. Korth, S. Sudarshan, (2010), Database System Concepts, (Sixth Edition), McGraw-Hill

Baazizi, M. A., Colazzo, D., Ghelli, G., Sartiani, C., & Scherzinger, S. (2023). Negation-closure for JSON Schema. \*Theoretical Computer Science\*, 955, 113823.

Banzal, S. (2017). \*Learning XML\*. Laxmi Publications Pvt Limited.

Basso, M., Schiavio, F., Rosà, A., & Binder, W. (2022, March). Optimizing Parallel Java Streams. In 2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS) (pp. 23-32). IEEE.

Berg, K. L., Seymour, T., & Goel, R. (2013). History of databases. \*International Journal of Management & Information Systems (Online)\*, 17(1), 29.

Bercich, N. (2002). “The Evolution of the Computerized Database”, \*Concept\*, 26. Retrieved from [link missing].

Bikaki, A., Peters, M., Krozel, J., & Kakadiaris, I. A. (2024). Building an open-source collaborative platform for migration research: A metadata modeling approach using XML. \*Knowledge-Based Systems\*, 299, 111823.

Boisot, M., & Canals, A. (2004). Data, information and knowledge: have we got it right?. Journal of evolutionary economics, 14, 43-67.

Bozic, R. (2022). Advantages and challenges of NoSQL compared to SQL databases – A systematic literature review. Journal of Faculty of Economics Brcko, Volume 16, Issue 1, 2022, pp. 11-20.

Bruckner, F. (2012). Wide Column Stores. Hochschule Mannheim.

Chandra, D. G. (2015). BASE analysis of NoSQL database. \*Future Generation Computer Systems\*, 52, 13-21.

Chen, J. K., & Lee, W. Z. (2019). An Introduction of NoSQL Databases based on their categories and application industries. \*Algorithms\*, 12(5), 106.

Chan, Y., Wellings, A., Gray, I., & Audsley, N. (2017). A distributed stream library for Java 8. IEEE Transactions on Big Data, 3(3), 262-275.

Cheng, G., Ying, S., Wang, B., & Li, Y. (2021). Efficient performance prediction for apache spark. Journal of Parallel and Distributed Computing, 149, 40-51.

Chillón, A. H., Klettke, M., Ruiz, D. S., & Molina, J. G. (2024). A Generic Schema Evolution Approach for NoSQL and Relational Databases. \*IEEE Transactions on Knowledge and Data Engineering\*.

Clifton, C., & Garcie-Molina, H. (2000, January). The design of a document database. In \*Proceedings of the ACM conference on Document processing systems\* (pp. 125-134).

Chua, K. M., & Lee, W. Z. (2019). An Introduction to NoSQL Databases based on their categories and application industries. \*Algorithms\*, 12(5), 106.

Date, C. J. (2003). \*An Introduction to Database Systems\*, Fifth Edition. Boston, MA: Addison Wesley.

Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. ACM Computing Surveys (CSUR), 51(2), 1-43.

Dey, D., Barron, T. M., & Storey, V. C. (1996). A complete temporal relational algebra. \*The VLDB journal\*, 5, 167-180.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., & Suciu, D. (1998). XML-QL: a query language for XML.

Dieu, G., Lerat, J. S., & Cremer, S. (2023, May). Relational Wide Column Store: a new Big Data paradigm. In \*Proceedings of the 20th ACM International Conference on Computing Frontiers\* (pp. 211-212).

DuCharme, B. (2012, January 26). 25 years of database history (starting in 1955). [Web blog post]. Retrieved from http://www.snee.com/bobdc.blog/

Eessaar, E. (2016). The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground. \*Baltic Journal of Modern Computing\*, 4(1).

Fireship. (n.d.). Top seven database paradigms. Fireship.io. <https://fireship.io/lessons/top-seven-database-paradigms/>

Gonzalez, J. E. (2014, April). From graphs to tables the design of scalable systems for graph analytics. In WWW (Companion Volume) (pp. 1149-1150).

Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., & Saxena, U. (2017, October). NoSQL databases: Critical analysis and comparison. In *2017 International conference on computing and communication technologies for smart nation (IC3TSN)* (pp. 293-299). IEEE.

Győrödi, C., Győrödi, R., Pecherle, G., & Olah, A. (2015, June). A comparative study: MongoDB vs. MySQL. In \*2015 13th international conference on engineering of modern electric systems (EMES)\* (pp. 1-6). IEEE.

Han, J., Haihong, E., Le, G., & Du, J. (2011, October). Survey on NoSQL database. In \*2011 6th international conference on pervasive computing and applications\* (pp. 363-366). IEEE.

Harrington, J. L. (2016). \*Relational database design and implementation\*. Morgan Kaufmann.

Hector Garcia-Moilna, Jeffrey D, Ullman, Jennifer Widom, (2008), Database Systems: The Complete Book, (2nd Edition) Department of Computer Science Stanford University; PEARSON, Prentice Hall.

Hurst, N. (n.d.). Visual guide to NoSQL systems. Nathan Hurst's Blog. https://blog.nahurst.com/visual-guide-to-nosql-systems

Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). Learning spark: lightning-fast big data analysis. "O'Reilly Media, Inc."

Kathare, N., Reddy, O. V., & Prabhu, V. (2020). A comprehensive study of Elasticsearch. \*International Journal of Science and Research (IJSR)\*.

Khan, W., Kumar, T., Zhang, C., Raj, K., Roy, A. M., & Luo, B. (2023). SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review. Big Data and Cognitive Computing, 7(2), 97

|  |
| --- |
|  |
| Khodorovskii, V. V. (2002). On normalization of relations in relational databases. *Programming and Computer Software*, *28*, 41-52. |
|  |

Kononenko, O., Baysal, O., Holmes, R., & Godfrey, M. W. (2014, May). Mining modern repositories with elasticsearch. In \*Proceedings of the 11th working conference on mining software repositories\* (pp. 328-331).

Kroenke, D. & Auer, D. (2007). \*Database Concepts\*. 3rd ed. New York, NY: Prentice.

Lausen, G., Meier, M., & Schmidt, M. (2008, March). SPARQLing constraints for RDF. In \*Proceedings of the 11th international conference on Extending database technology: Advances in database technology\* (pp. 499-509).

Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. \*ACM SIGOPS operating systems review\*, 44(2), 35-40.

Malik, U., Goldwasser, M., & Johnston, B. (2019). \*SQL for Data Analytics: Perform fast and efficient data analysis with the power of SQL\*. Packt Publishing Ltd.

Mehta, D., Suhagiya, H., Gandhi, H., Jha, M., Kanani, P., & Kore, A. (2023). SQLIML: A comprehensive analysis for SQL injection detection using multiple supervised and unsupervised learning schemes. SN Computer Science, 4(3), 281.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Talwalkar, A. (2016). Mllib: Machine learning in apache spark. Journal of Machine Learning Research, 17(34), 1-7.

Mihai, G. (2020). Comparison between relational and NoSQL databases. \*Econ. Appl. Inform\*, 3, 38-42.

Miller, J. J. (2013, March). Graph database applications and concepts with Neo4j. In \*Proceedings of the southern association for information systems conference\*, Atlanta, GA, USA (Vol. 2324, No. 36, pp. 141-147).

Mongo, D. B. (2015). \*Mongodb\*.

Osemwegie, O., Okokpujie, K., Nkordeh, N., Ndujiuba, C., John, S., & Stanley, U. (2018). Performance benchmarking of key-value store NoSQL databases. \*International Journal of Electrical and Computer Engineering (IJECE)\*, 8(6), 5333-5341.

Petković, D. (2017). SQL/JSON standard: properties and deficiencies. *Datenbank-Spektrum*, *17*(3), 277-287.

Puangsaijai, W., & Puntheeranurak, S. (2017, March). A comparative study of relational database and key-value database for big data applications. In \*2017 International Electrical Engineering Congress (iEECON)\* (pp. 1-4). IEEE.

Rosales, E., Rosà, A., Basso, M., Villazón, A., Orellana, A., Zenteno, Á., ... & Binder, W. (2022, March). Characterizing Java Streams in the Wild. In 2022 26th International Conference on Engineering of Complex Computer Systems (ICECCS) (pp. 143-152). IEEE.

Ryza, S., Laserson, U., Owen, S., & Wills, J. (2017). Advanced analytics with spark: patterns for learning from data at scale. "O'Reilly Media, Inc.".

Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. International Journal of Data Science and Analytics, 1, 145-164.

Seghier, N. B., & Kazar, O. (2021, September). Performance benchmarking and comparison of NoSQL databases: Redis vs mongodb vs Cassandra using YCSB tool. In \*2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)\* (pp. 1-6). IEEE.

Sholichah, R. J., Imrona, M., & Alamsyah, A. (2020, September). Performance analysis of neo4j and mysql databases using public policies decision making data. In \*2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)\* (pp. 152-157). IEEE.

Taylor, A. (2000). \*Database Development for Dummies\*. New York, NY: John Wiley & Sons, Inc.

Taylor, A. (2007). \*SQL for Dummies\*. New York, NY: John Wiley & Sons, Inc.

Tiwari, S. (2011). *Professional nosql*. John Wiley & Sons.

University of Rochester. (n.d.). \*Term paper on NoSQL\*. https://www.cs.rochester.edu/courses/261/spring2017/termpaper/06/Termpaper\_NoSQL\_paper.pdf

Vera, H., Boaventura, W., Holanda, M., Guimaraes, V., & Hondo, F. (2015, September). Data modeling for NoSQL document-oriented databases. In \*CEUR Workshop Proceedings\* (Vol. 1478, pp. 129-135).

Vidal, X. C., & Manzano, R. C. (2014). Taking a closer look at LHC. [online] Available: <http://www.lhc-closer.es/1/3/12/0>.

Xin, R. S., Gonzalez, J. E., Franklin, M. J., & Stoica, I. (2013, June). Graphx: A resilient distributed graph system on spark. In First international workshop on graph data management experiences and systems (pp. 1-6).

Zaharia, M. (2016). An architecture for fast and general data processing on large clusters. Morgan & Claypool.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2014). Resilient distributed datasets. In A fault-tolerant abstraction for in-memory cluster computing in Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In 2nd USENIX workshop on hot topics in cloud computing (HotCloud 10).

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013, November). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the twenty-fourth ACM symposium on operating systems principles (pp. 423-438).

Zhang, N., & Huang, F. (2016). An Introduction to NoSQL Databases and Applications. Retrieved from

Zhao, L. (2020). *An Introduction to NoSQL Databases and Their Applications*. International Journal of Computer Science