# מעבר ל-SQL מול NoSQL: אפליקציות מודרניות על Aurora DSQL

AWS Summit Tel Aviv 2026 ,סיכום שנן — TLV-DAT306

Pini Dibask, Principal Database Solution Architect, AWS | Eran Mhadipor, Senior Solution Architect, AWS

10 בספטמבר 2026 | משך: כ-43 דקות | הופק מהקלטת הסשן

מסלול: Databases on AWS | רמה: 300

# על המסמך הזה

המסמך מסכם את הסשן DAT306 מתוך AWS Summit Tel Aviv 2026, בהתבסס על ההקלטה המלאה. הוא נבנה מתמלול אוטומטי של ההקלטה ומהסליידים שחולצו מהווידאו עצמו, כך שאפשר לחזור לתוכן בלי לצפות שוב בארבעים ושלוש הדקות. חותמות הזמן בגוף המסמך מפנות לנקודה המדויקת בהקלטה.

הסשן מחולק לשני חלקים ברורים: החלק הראשון (עד דקה 21 בערך) מוקדש למנגנון הפנימי של Aurora DSQL ולדמו סקייל, והחלק השני לחוויית ההקמה, לארכיטקטורת רפרנס serverless, לעלות ולחיבור סוכני AI דרך MCP.

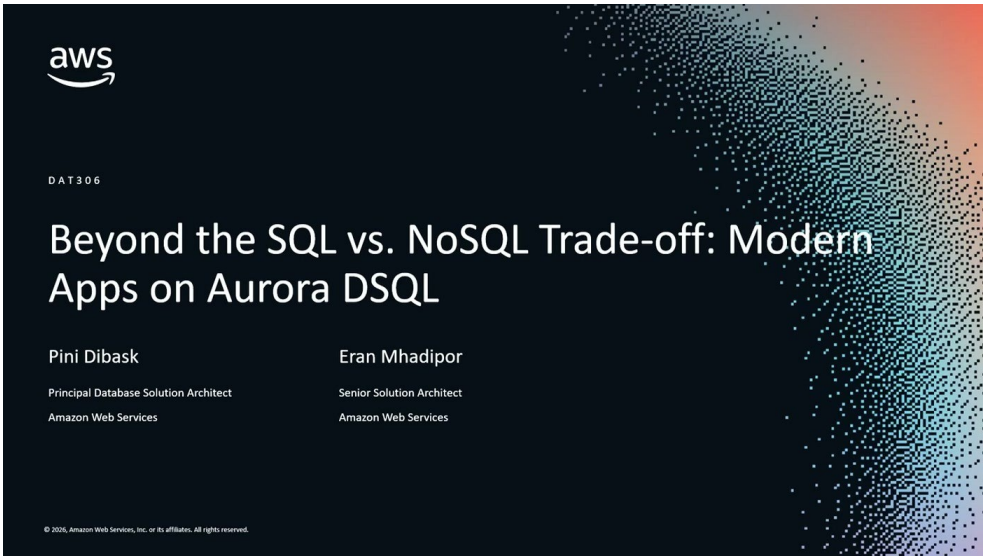
### איך לקרוא את זה

- **פרק 1 — תקציר מנהלים:** חמש דקות קריאה. מה הוצג ומה משמעותי לצוותי דאטה ופיתוח.
- **פרקים 2–5 — המנגנון:** למה נבנה שירות חדש, הטופולוגיות, הרכיבים הפנימיים והגבולות התפעוליים.
- **פרק 6 — הדמו:** מספרי הסקייל שהוצגו על הבמה ומה בדיוק רץ מאחוריהם.
- **פרקים 7–10 — העבודה בפועל:** הקמה ותפעול, ארכיטקטורת serverless, מודל העלות וחיבור סוכני AI.
- **פרק 11 + סיכום:** מתי לבחור בשירות ומה לוקחים מכאן.

**הערת מקור** המסמך הופק מתמלול אוטומטי של ההקלטה. התמלול מדויק בקטעים המקצועיים, אך מונחים לועזיים מופיעים בו לעיתים בשיבוש פונטי (למשל "מזהרנו" במקום "מיזערנו", "Walkman" במקום working memory). בגוף המסמך המונחים מנורמלים לצורתם האנגלית, ואילו הציטוטים מובאים כלשונם בתמלול ואומתו מול חותמת הזמן המצוינת. מספרים שמופיעים על הסליידים צוינו ככאלה.

# 1. תקציר מנהלים

- **Aurora DSQL הוא ניסיון לסגור פער ספציפי, לא להחליף את כל הפורטפוליו:** בסיס נתונים רלציוני תואם PostgreSQL עם ACID מלא ו-strong consistency, שמתנהג תפעולית כמו DynamoDB — serverless, ללא אינסטנסים וללא maintenance window. הדובר הראשון פתח בכך שהשירות יצא ל-GA "שנה שעברה" [0:00].
- **שתי הבעיות שהוא בא לפתור הן סקייל של כתיבות ועומס תפעולי:** ב-RDS/Aurora אפשר 15 read replicas, אבל "בסוף writer node, ה-head node של הכתיבות הוא אחד" [1:37]; ולצד זה vacuum, data files, שדרוגי גרסה וחלונות תחזוקה שנעלמים כאן.
- **המחיר של strong consistency שולם רק בזמן ה-commit:** בזכות optimistic concurrency control אין round-trip חוצה-ריג'ן לכל פקודה אלא רק בסיום הטרנזקציה, ולכן משתלם לארוז מאות statements בטרנזקציה אחת. read-only transactions נמדדו ב-single digit millisecond ב-p99 [7:47].
- **הדמו הראה מספרים שלא מקובלים בעולם הרלציוני:** 50 קונטיינרים ב-100 × ECS Fargate קונקשנים = 5,000 חיבורים, יותר מ-380K טרנזקציות לשנייה, יותר מ-3.8M row operations לשנייה ומעל 2 מיליארד פעולות מצטברות בכעשר דקות (מספרי הסלייד Demo Summary), בעומס של 80% כתיבות.
- **חוויית ההקמה היא הטענה המרכזית של החלק השני:** קלאסטר single-region בפחות מדקה וקלאסטר multi-region active-active בפחות משלוש וחצי דקות, בלי בחירת אינסטנסים, subnets או security groups [27:44].
- **הגבולות התפעוליים הם חלק מהעיצוב ולא באג:** 5 דקות timeout לטרנזקציה, 128MB working memory לשאילתה, 3,000 שורות שניתן לשנות בטרנזקציה, ושעה כ-timeout לחיבור — מי שלא מתכנן לפיהם ייתקל בהם בפרודקשן.




*סלייד הפתיחה: קוד הסשן, הכותרת ושני הדוברים — Pini Dibask ו-Eran Mhadipor.*

# 2. למה AWS הוסיפה עוד בסיס נתונים

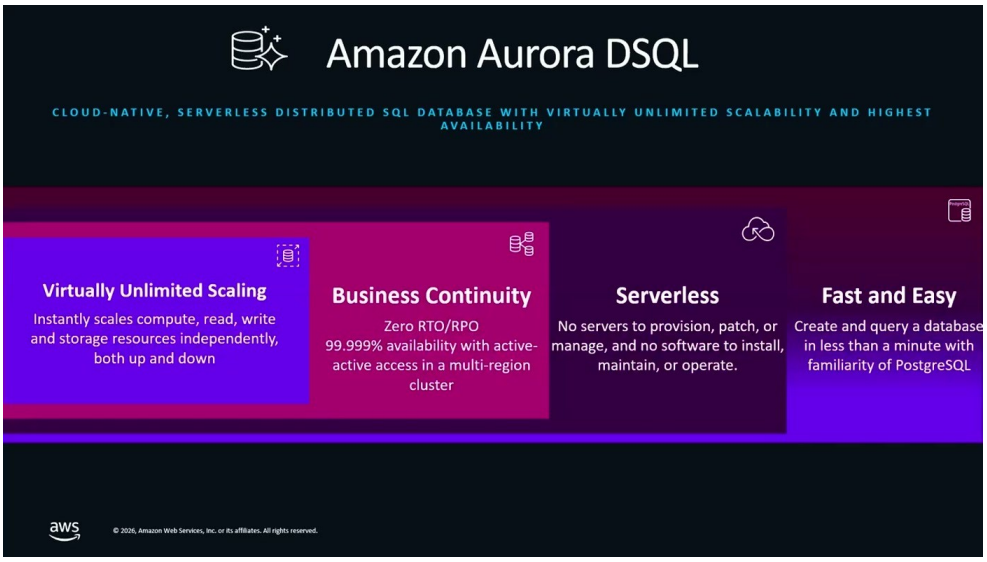
הפתיחה מציגה את הפורטפוליו הרחב של שירותי הדאטהבייס ושואלת ישירות למה צריך עוד אחד. התשובה מתמקדת בשני כאבים שחוזרים מלקוחות: סקייל של כתיבות ועומס תפעולי.

על סקייל — גם read replicas רבים לא פותרים את צוואר הבקבוק של הכתיבה, כי הכתיבות מרוכזות בצומת אחד. גם אינסטנס ענק, בדוגמה שניתנה R8g בגודל 48xlarge עם 192 vCPU, נשאר אינסטנס בודד עם תקרה משלו [1:37].

> **Solution Architect, AWS [1:37] —** בסוף writer node, ה-head node של הכתיבות הוא אחד

על תפעול — הדובר מספר על שיחה עם DBA לפני ההרצאה בנוגע ל-vacuum ו-data files ב-PostgreSQL, ומכריז שב-Aurora DSQL אין vacuum, אין ניהול data files, ואין שדרוגי גרסה עם downtime.

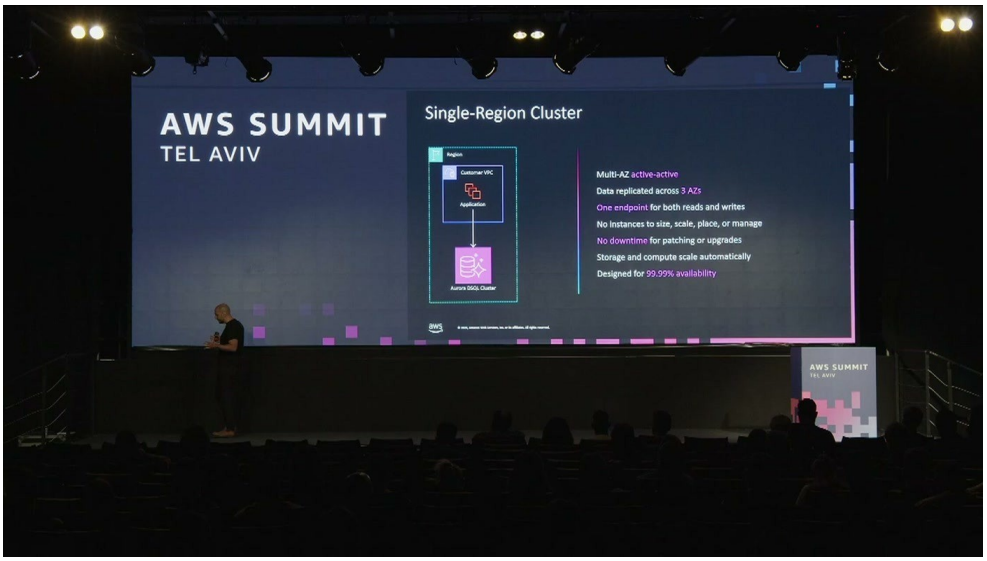
> **[1:37] —** לא צריך לתעסק עם שדרוגי גרסאות ודאונטיים, אין דבר כזה מיינטנס ווינדו




*ארבע ההבטחות שעליהן נשען השירות: Virtually Unlimited Scaling, Business Continuity עם Zero RTO/RPO ו-99.999% זמינות בקלאסטר multi-region, Serverless, ו-Fast and Easy.*

נקודה שהודגשה כייחודית לעולם הרלציוני היא Zero RTO ו-Zero RPO — בין אם בתצורת single-region ובין אם ב-multi-region [3:09]. בפתרונות המסורתיים, נאמר, אין היום מענה כזה.

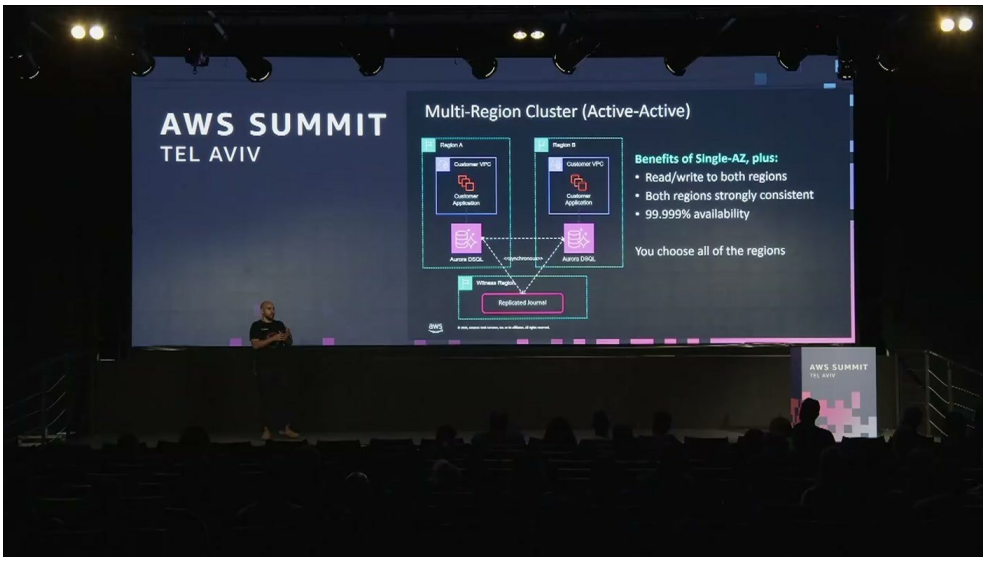
## 3. שתי הטופולוגיות: Single-Region ו-Multi-Region




*הקונפיגורציה הבסיסית: endpoint יחיד, נתונים המשוכפלים על שלושה Availability Zones, ללא אינסטנסים לניהול ו-99.99% זמינות מתוכננת.*

בתצורת single-region אין הפרדה בין endpoint לקריאה ל-endpoint לכתיבה, ולכן גם לא צריך read/write splitting בשכבת האפליקציה. הסקיילינג של הרכיבים הפנימיים מתבצע דינמית מאחורי הקלעים לפי ה-workload.

> **[3:09] —** אין הפרדה של Endpoints ל-rights ול-reads. יש לכם Endpoint אחד, אתם לא צריכים לחשוב על Read-rights-splitting




*התצורה החוצה-ריג'נית: שני ריג'נים אקטיביים עם endpoint לכל אחד, ריג'ן witness שלישי שמחזיק Replicated Journal, ו-99.999% זמינות.*

ב-multi-region אין primary ו-secondary ואין leader ו-follower — שני הריג'נים אקטיביים, ואפשר לקרוא ולכתוב לשניהם במקביל תוך שמירה על strong consistency.

> **[4:40] —** יש לכם שני ריג'נים, שניהם אקטיביים, שניהם למעלה, אתם יכולים לפנות לשניהם בו זמנית, ומובטח לכם strong consistency

הריג'ן השלישי בתרשים הוא witness region: אין לו endpoint, אי אפשר לקרוא או לכתוב ממנו, והוא מחזיק עותק של ה-journal בלבד. תפקידו לשמש tiebreaker לקוורום, למשל כשריג'ן אחד נופל [4:40]. האבחנה המעשית: אם בוצע commit בריג'ן A, שאילתה שתרוץ בריג'ן B מיד לאחר מכן תראה את השינוי.

— על ה-[4:40] witness region הוא בסך הכל מכיל replicated journal

# 4. מה קורה מתחת למכסה המנוע

## 4.1 היכן משולם ה-latency

השאלה שחוזרת מלקוחות היא איך שומרים על strong consistency בין ריג'נים בלי לשלם על כך בכל פקודה. התשובה שניתנה: הרפליקציה סינכרונית, אבל התיאום החוצה-ריג'ני מרוכז כולו לרגע ה-commit.

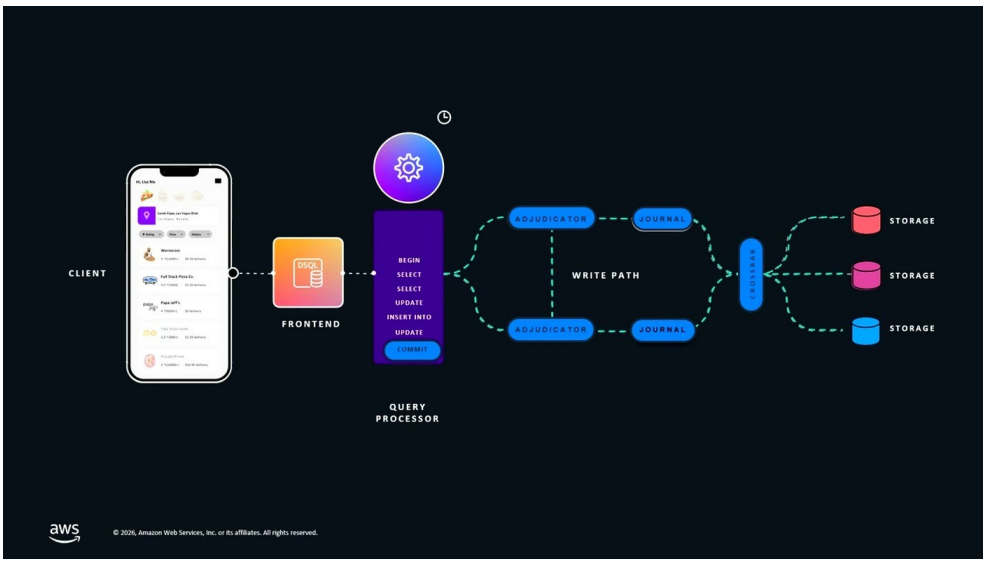
> **— [6:17]** אנחנו מזהרנו את ה-latency, אך ורק לזמן הקומית

המסקנה התכנונית שנגזרת מכך הפוכה מהאינטואיציה של רבים: עדיף לארוז מאות או אלפי statements בטרנזקציה אחת ולא לפזר אותם לטרנזקציות קצרות, כי כל טרנזקציה משלמת round-trip אחד ולא אחד לכל פקודה [7:47]. טרנזקציות read-only לא דורשות תיאום עם הריג'ן השני כלל.

> **— [7:47]** read-only transactions, מאוד מאוד פרנזקציות מהירות, single digit millisecond latency, באחוזון 99

בסלייד המדידות הוצגו זמני p99 ל-SELECT, UPDATE ו-COMMIT ב-single region וב-multi region. לפי הדובר, הערכים הם single digit millisecond בכל המקרים למעט אחד: זמן ה-commit בתצורת multi-region, שם הם עולים ל-double digit millisecond ותלויים במרחק בין הריג'נים שנבחרו [7:47].

## 4.2 הרכיבים הפנימיים




*מסלול הכתיבה: מהקליינט דרך ה-frontend ל-Query Processor שמריץ את בלוק הטרנזקציה, ומשם ל-Adjudicator, ל-Journal ודרך ה-crossbar לשכבות ה-storage.*

- **Transaction Router:** מקבל את החיבור ומייצר את הקונקשן.
- **Query Processor (QP):** מיקרו-VM שמריץ PostgreSQL — הרכיב היחיד בארכיטקטורה שמבצע parsing, planning ו-execution.
- **Adjudicator:** "השופט" — אחראי על ה-Isolation שב-ACID, ומכריע בזמן ה-commit אם יש התנגשות עם טרנזקציה אחרת שנגעה באותן שורות.
- **Journal:** distributed commit log שאחראי על ה-Durability. ברגע שהמידע בו, הטרנזקציה דורבילית.
- **Storage:** ה-NVMe SSD שמחזיקים את הרשומות עצמן.

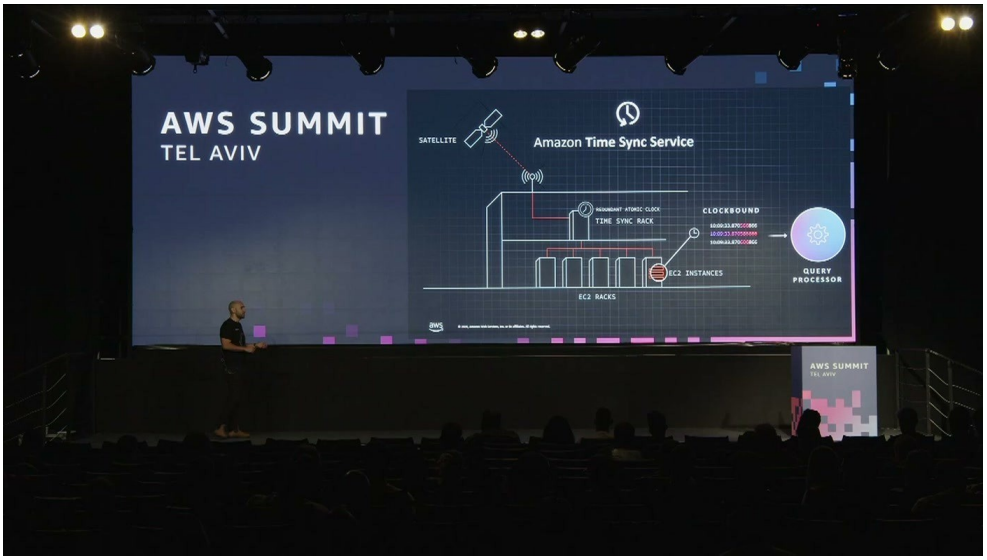
התוצאה של הפרדת ה-Transaction Router מה-QP היא שאין צורך ב-server-side connection pooling — לא RDS Proxy ולא PgBouncer. המודל המנטלי שהוצע שונה מהותית מאינסטנס PostgreSQL גדול יחיד.

> **— [9:18]** תחשבו על מאות או אלפי פוסטגרסים קטנים, מיקרוביימים קטנים, שכל אחד מהם זה קונקשן שמריץ פוסטגרס

### 4.3 נעילות, deadlocks וזמן

מכיוון שבדיקת ההתנגשויות מתרחשת רק בסיום הטרנזקציה, אין נעילות במהלכה — ולכן, לפי הדובר, אין deadlocks.

> **— [10:52]** דדלוק זה לא בעיה דטאביסית, זה בעיה אפליקטיבית, היא לא קיימת באור ה-DSQL, פשוט מהסיבה שאין לנו נעילות בזמן טרנזקציה




*Amazon Time Sync Service: שעונים אטומיים וקליטת GPS ברמת ה-rack, שמספקים ל-Query Processor חלון זמן מדויק (clockbound).*

במקום Transaction ID כמו ב-MVCC של PostgreSQL, המנגנון כאן מבוסס זמן. Amazon Time Sync Service מספק דיוק ברמת ננו-שניות, ולכן ידוע בדיוק מתי טרנזקציה התחילה ומתי הסתיימה.

> **— על Time Sync Service [12:24]** שמאחורי הקלעים משתמש בשעונים אטומיים ולאביני GPS

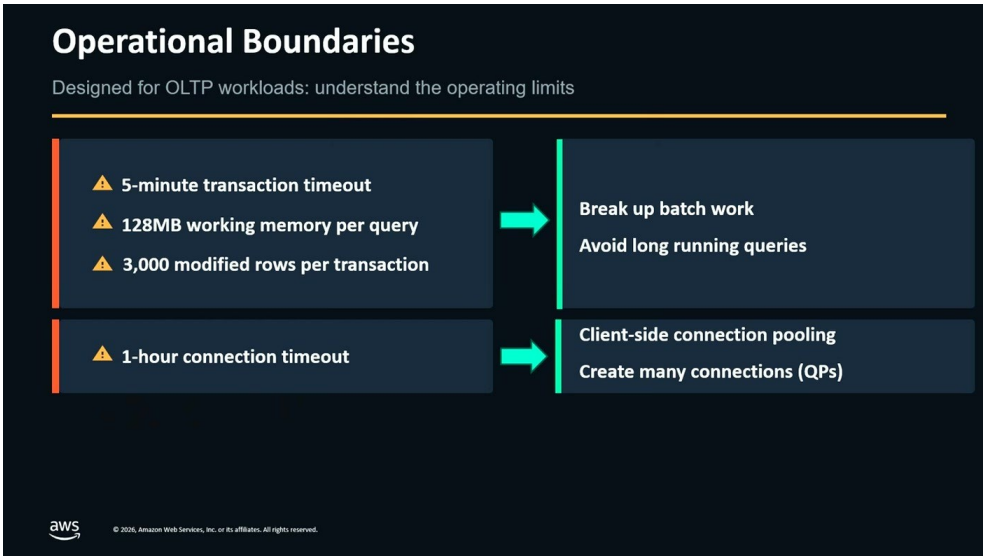
רמת הבידוד שמתקבלת היא snapshot isolation — חזקה מברירת המחדל של PostgreSQL, שהיא read committed. הדימוי שניתן: הטרנזקציה מרגישה כאילו היא לבדה בעולם, ושני SELECT באותה טרנזקציה לא יחזירו תמונות שונות [13:56].

בטרנזקציות קריאה בלבד הגישה היא ישירות ל-storage, בלי מעבר ב-Adjudicator וב-Journal. בטרנזקציות כתיבה המעבר דרכם הכרחי, וכאן מגיע הכלל שהאפליקציה חייבת להפנים: המנצחת היא הטרנזקציה הראשונה שביצעה commit — לא הראשונה שנפתחה — והשנייה תקבל שגיאה ותידרש ל-retry.

> **— [13:56]** שימו לב שהאג'ודיקטור מוודא שהטרנזקציה הראשונה שביצע קומית היא תנצח

# 5. גבולות תפעוליים

לפני הדמו הוצגו מה שהדובר מכנה "הגבולות התפעוליים" — החלטות ארכיטקטוניות מכוונות שנועדו להבטיח scaling צפוי, ולא מגבלות זמניות. השירות נבנה ל-OLTP: טרנזקציות קצרות ומהירות.




*ארבעת הגבולות והפעולה שכל אחד מהם מחייב: פיצול batch לחלקים, הימנעות משאילתות ארוכות, connection pooling בצד הקליינט ויצירת חיבורים רבים.*

**[15:29]** — 5 דקות Transaction Timeout, 128 מגה בייט עבור ה-Walkman בפוסטגרס, ועד 3000 שורות שאנחנו יכולים לשנות בטרנזקציה

החיבורים מוגבלים לשעה כדי לאפשר החלפת רכיבים פנימיים בצורה שקופה. ההמלצה המעשית: להגדיר בצד הקליינט connection pool עם max lifetime של כ-55 דקות, כך שהחיבורים יתחדשו ביוזמת האפליקציה ולא ייסגרו על ידי השירות [15:29].

הכלל האחרון הוא הכלל שמגיע מעולם ה-NoSQL: מכיוון שהקונפליקטים נבדקים בזמן commit, צריך להימנע מ-hot write keys — בדיוק כפי שנמנעים מ-hot partition key ב-DynamoDB.

**[17:01]** — אני אעדיף להשתמש ב-UID v4 כדי למזהר את הקונפליקטים בזמן קומית

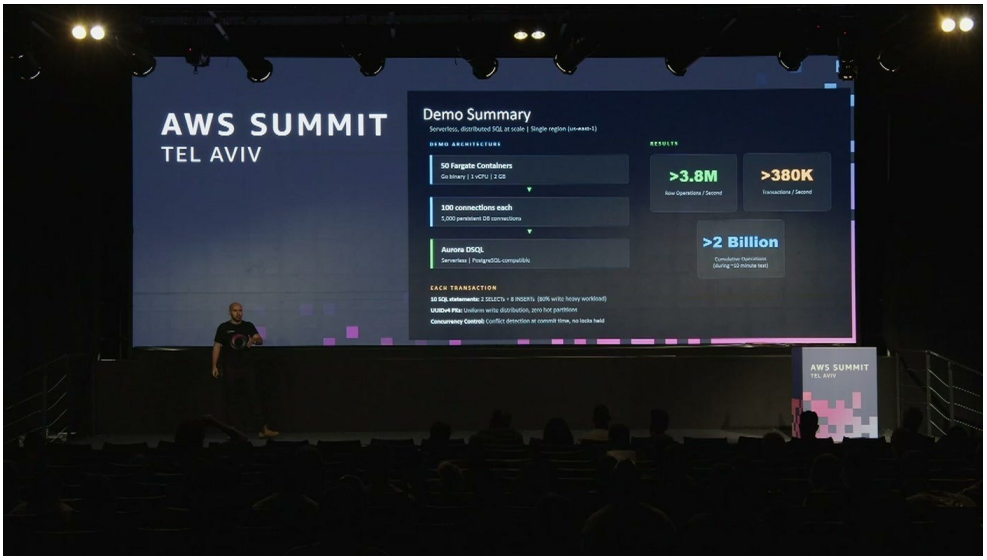
# 6. הדמו: מה באמת רץ ומה יצא

הדמו הורכב מקלאסטר single-region ומ-load generator שרץ על Amazon ECS Fargate. כל קונטיינר הריץ 100 חיבורים, וכל חיבור לולאה אינסופית של טרנזקציות שבכל אחת 10 פקודות — שני SELECT ושמונה כתיבות, כלומר 80% כתיבות. המפתחות נוצרו כ-UUID כדי לפזר את הכתיבות ולמנוע התנגשויות [18:32].

החיבור עצמו מדגים נקודה תפעולית נוספת — אין סודות קבועים לניהול.

> **[18:32]** מתחברים אליו באמצעות טוקן, אין דבר כזה user password שצריך לשמור

השלב הראשון רץ עם 25 קונטיינרים, כלומר 2,500 חיבורים, והגיע לכ-200 אלף טרנזקציות לשנייה. הכפלת מספר הקונטיינרים ל-50 (5,000 חיבורים) הכפילה את התפוקה לכ-400 אלף טרנזקציות לשנייה — הנקודה שהדובר רצה להראות היא הסקייל הליניארי [20:03]. תקרת החיבורים היא 10,000 והיא soft limit שניתן להגדיל בבקשה [18:32].




*סיכום הדמו כפי שהוצג: 50 קונטיינרים, 5,000 חיבורים, מעל 3.8 מיליון row operations לשנייה, מעל 380 אלף טרנזקציות לשנייה ומעל 2 מיליארד פעולות מצטברות בכעשר דקות.*

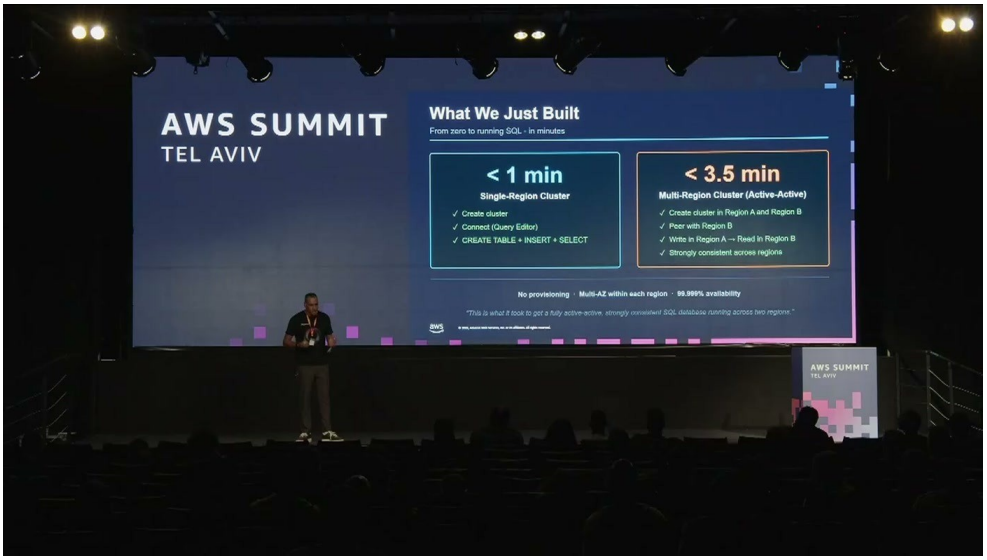
> **[20:03]** הגענו למאות מיליוני טרנזקציות ושני מיליארד operations בעשר דקות של דמו

ההסתייגות נאמרה מפורשות: המספרים האלה מותנים בעבודה נכונה עם מפתחות מפוזרים.

> **[21:35]** אם אתם עובדים נכון עם UUID, ואתם מזרים את הקונפליקטים בזמן הקומית, אז אתם יכולים להגיע באמת לסקל שהוא מאוד משמעותי

# 7. הקמה, תפעול ו-observability

החלק השני הועבר לדובר השני, שלפי ההצגה שימש בעבר ראש צוות הפיתוח של Aurora DSQL [21:35]. הטיעון שלו: מול רוב שירותי בסיסי הנתונים בענן, עוד לפני שאילתה ראשונה נדרשות החלטות על סוגי אינסטנסים וגודלם, על Availability Zones, על security groups ועל subnets — ולעיתים כמה סבבים עד שמגיעים לקונפיגורציה אופטימלית מבחינת עלות וביצועים [23:07].

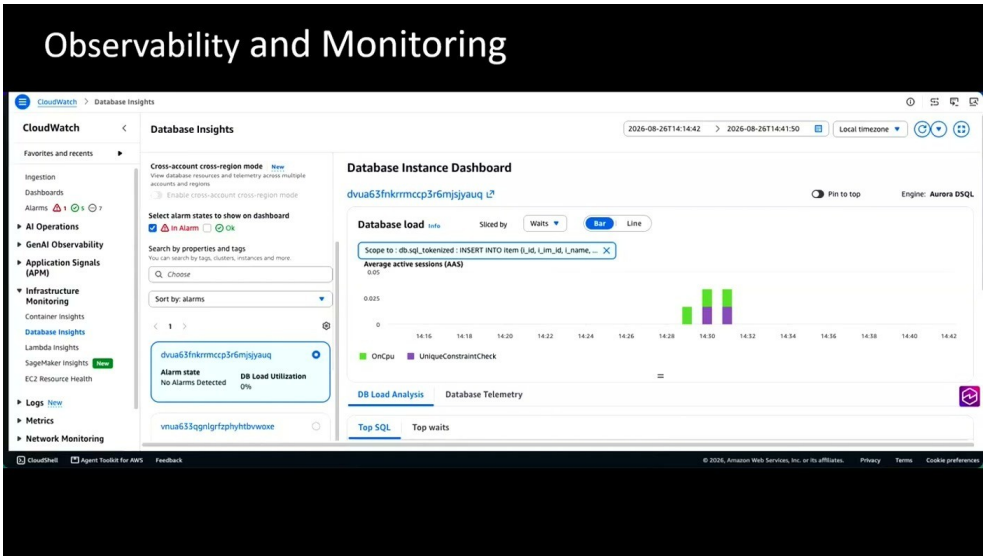



*סיכום שני הדמואים של ההקמה: קלאסטר single-region בפחות מדקה וקלאסטר multi-region active-active בפחות משלוש וחצי דקות, כולל peering וכתיבה בריג'ן אחד וקריאה בשני.*

בדמו הראשון הוקם קלאסטר single-region בכמה קליקים, ומרגע שהסטטוס Active אפשר להתחבר דרך ה-query editor המובנה בקונסול, ליצור סכמה וטבלאות, להכניס data ולהריץ שאילתות — כולל EXPLAIN ANALYZE [23:07]. בדמו השני הוקמו שני קלאסטרים, אחד ב-US East ואחד ב-US West, וחוברו בתהליך peering שמונחה צעד-אחר-צעד בקונסול.

**— על הקמת קלאסטר multi-region [27:44]** וכל זה לקח לנו פחות משלוש וחצי דקות

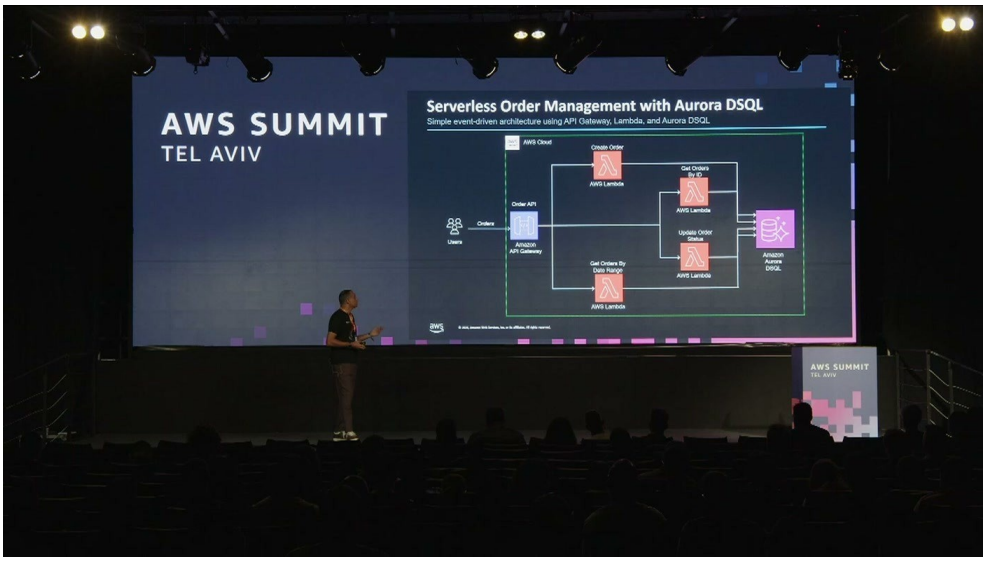
**— [27:44]** אבל מאוד חשוב שתראו מה לא עשינו לא הגדרנו שום קונפיגורציה מסובכת לא בחרנו אינסטנסים, לא הגדרנו נטוורקינג




*מסך CloudWatch Database Insights עם Engine: Aurora DSQL — עומס הדאטהבייס לפי wait events, Top SQL, ושאילתת INSERT שמזוהה כ-UniqueConstraintCheck.*

בצד ה-observability, הקונסול חושף את המדדים המרכזיים — מספר קונקשנים, query timeouts, קונפליקטים של OCC — וכולם זמינים גם ב-Amazon CloudWatch, לצד usage metrics למעקב עלות. הדובר ציין ש"ממש לפני מספר שבועות" צוות השירות הוסיף את Aurora DSQL גם ל-CloudWatch Database Insights, שאוסף מטריקות ברזולוציה של שנייה מכלל המיקרו-שירותים של DSQL ומאפשר לזהות wait events ו-Top SQL [24:38].

## 8. ארכיטקטורת רפרנס: מערכת הזמנות serverless




*Amazon API Gateway מול ארבע פונקציות Lambda — יצירת הזמנה, עדכון סטטוס, שליפה לפי מזהה ושליפה לפי טווח תאריכים — שכולן פונות ישירות ל-Aurora DSQL.*

הדוגמה שנבחרה היא מערכת לניהול הזמנות: משתמשים ואפליקציות פונים ל-API Gateway, שמפעיל את פונקציית ה-Lambda המתאימה לפי הפעולה, וכל פונקציה מתחברת ישירות ל-Aurora DSQL. היתרון המבני — אין server-side connection pooling, וכל Lambda עושה scale באופן עצמאי מהאחרות בזמן ש-DSQL עושה scale כדי לענות לכולן [29:20].

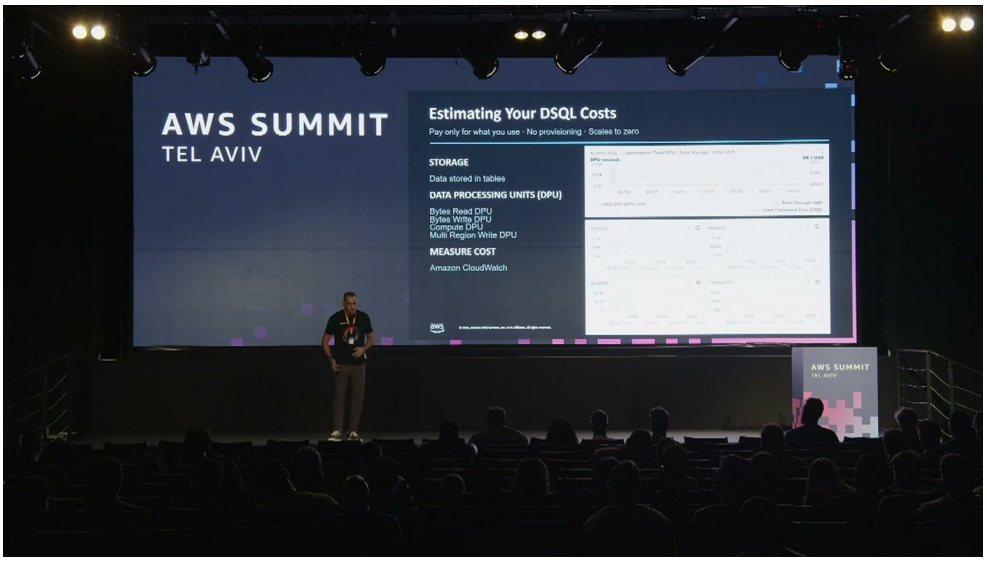
> **— [29:20]** זה שאם עכשיו אין לנו שום קריאות API כל הארכיטקטורה הזאת יורדת לאפס, כמעט לאפס

הפרופיל שהוגדר כמתאים לשירות מסוכם בשלושה מאפיינים: טרנזקציות קצרות, serverless מקצה לקצה, ואלפי חיבורים במקביל [30:54].

המעבר ל-multi-region לא משנה את קוד האפליקציה: מקימים קלאסטר שני, מבצעים peering, פורסים את אותן Lambda ואת API Gateway בריג'ן השני, ומנתבים משתמשים לריג'ן הקרוב באמצעות Amazon Route 53.

> **— [30:54]** כל למדה בכל ריג'ן מסתכלת אך ורק על הקלאסטר אינד פוינט של הריג'ן המקומי

# 9. מודל העלות




*שני מרכיבי החיוב — אחסון ו-DPU על ארבעת סוגיו — לצד מסכי המדידה ב-CloudWatch שמאפשרים לעקוב אחרי הצריכה בפועל.*

החיוב מחולק לשני חלקים. הראשון הוא storage — כמות הדאטה בטבלאות, שבקלאסטר multi-region נספרת לכל ריג'ן בנפרד. השני הוא יחידות עיבוד.

> **— [32:26]** והחלק השני זה Data Processing Units DPU

ה-DPU מודד את העבודה שהשירות ביצע: כמה בתים נקראו, כמה נכתבו, כמה compute נצרך לאורך הרכיבים השונים, ובתצורת multi-region גם כמה נכתב חוצה-ריג'נים. הסלייד מפרט את ארבעת הסוגים — Bytes Read DPU, Bytes Write DPU, Compute DPU ו-Multi Region Write DPU.

ההמלצה התפעולית: לעקוב אחרי ה-usage metrics ב-CloudWatch ולהגדיר alarms לזיהוי אנומליות ושינויים בצריכה. אבל האחריות המרכזית, נאמר במפורש, היא של המפתחים — תכנון סכמה נכון ואופטימיזציה של השאילתות [32:26]. מכאן עובר הסשן לכלים שאמורים לעזור בדיוק בזה.

# 10. סוכני AI ו-DSQL MCP Server

החלק האחרון עוסק בחיבור סוכני AI לבסיס הנתונים. השרשרת שהוצגה: אפליקציה או AI agent ← IDE (הדוגמאות שניתנו: Claude, Kiro, Codex או סוכן פנימי) ← Aurora DSQL ← DSQL MCP Server ← MCP client.

> **— [33:58]** וכל AI agent שיודע לעבוד עם MCP client יכול להתחבר לאור ה-DSQL דרך DSQL MCP Server

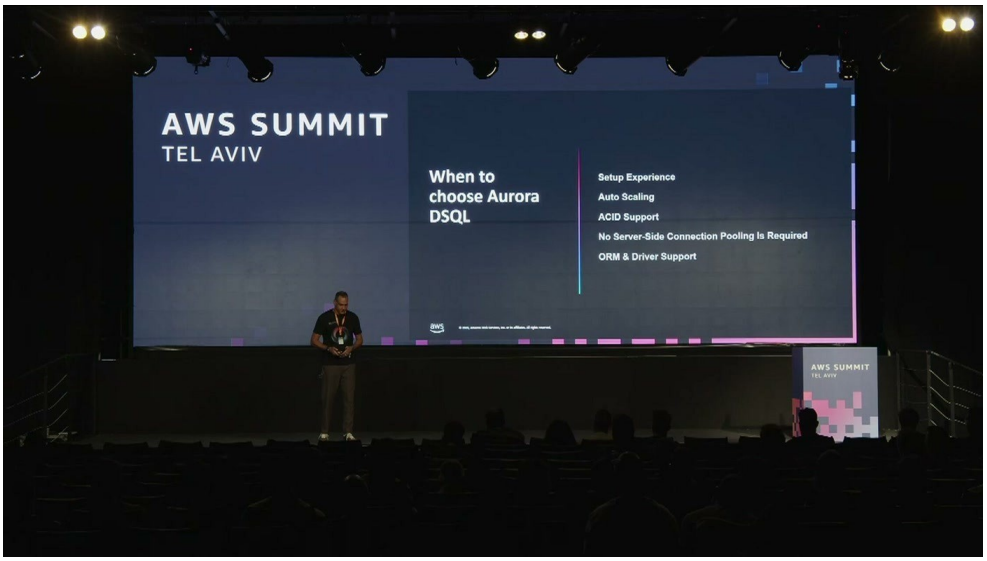
מה שה-MCP Server מוסיף מעבר לחיבור עצמו הוא הבנת הסכמה ו-skills — ידע מובנה על אופן העבודה הנכון מול השירות. ההזדהות היא ב-IAM ובטוקן, בהמשך ישיר לנקודה שאין ב-Aurora DSQL ניהול סיסמאות. ההגדרה עצמה דורשת שלושה פרמטרים בלבד: cluster endpoint, ה-region, ודגל שקובע אם לסוכן מותר גם לכתוב או שהוא מוגבל לקריאה בלבד [35:31].

### שלושת התרחישים שהודגמו

- **הבנת סכמה לא מוכרת:** במקום לחפש מסמכי ERD או אנשים בארגון שזוכרים מה עושה כל טבלה, הסוכן (Kiro בדוגמה) הריץ שאילתות דרך ה-MCP, החזיר את רשימת הסכמות והסביר את תפקידן, ואז ביצע drill-down לטבלאות, עמודות, קשרים ואינדקסים — בלי לצאת מה-IDE [35:31].
- **Business insights למשתמש לא טכני:** בקשה בשפה טבעית לאתר את הלקוחות עם ה-engagement הגבוה ביותר בקמפיין; הסוכן בנה שאילתה מותאמת לסכמה, הריץ אותה מול בסיס הנתונים והחזיר את הנתונים [37:06].
- **אופטימיזציה של SQL קיים:** הסוכן הריץ את השאילתה מול Aurora DSQL, ניתח את ה-EXPLAIN ANALYZE ואת ה-execution plan, הסביר מה לא תקין, כתב גרסה חדשה, הריץ אותה וזיהה את ההפרש בין השתיים [38:37].

> **הסתייגות מהבמה** גם בתרחיש האופטימיזציה הדובר עצר והבהיר שהפלט של הסוכן אינו מוצר מוגמר: את הקוד יש להעביר דרך מערכי הבדיקות ובדיקות הביצועים של הארגון, לוודא שלא נפגעה התנהגות עסקית, ורק אז לקדם לפרודקשן [38:37].

# 11. מתי לבחור ב-Aurora DSQL




*חמשת השיקולים שהוצגו כמצביעים לטובת השירות: חוויית ההקמה, auto scaling, תמיכת ACID, היעדר צורך ב-connection pooling בצד השרת, ותמיכה ב-ORM ובדרייברים.*

שלושת המסרים שהדובר ביקש להשאיר: השירות לא נועד להחליף כל בסיס נתונים קיים; צריך להכיר את ה-best practices שלו כדי להוציא ממנו את המרב; והוא שירות חדש שמתפתח מהר [38:37].

> **— [38:37]** הוא לא נועד להחליף כל שירות בסיס נתונים שאתם מכירים

על השאלה החוזרת — DSQL מול Aurora הרגיל, מול RDS, מול DynamoDB — התשובה שניתנה הייתה מפורשות לא חד-משמעית, ותלויה ב-workload, במאפייניו וביכולות ההנדסה והתפעול של הארגון.

> **— [40:09]** אז התשובה שלי זה שאין תשובה אחת נכונה זה מאוד תלוי בווקלוט שלכם ומאפיינים שלו

עם זאת ניתן כלל אצבע: מי שצריך SQL עם תמיכת ACID וזמינות גבוהה, ורוצה חוויית setup ו-auto scaling בסגנון DynamoDB — כדאי לו להסתכל על Aurora DSQL. נקודה נוספת שהופיעה כהורדת חיכוך: רוב ה-ORM והדרייברים שכבר נמצאים בקוד נתמכים, ולכן היקף השינויים בצד האפליקציה קטן [40:09].

## מה לוקחים מכאן

- **הטרנזקציה היא יחידת התכנון, לא השאילתה:** מכיוון שה-latency החוצה-ריג'ני משולם רק ב-commit, ארכיטקטורה שמפצלת עבודה להרבה טרנזקציות קטנות משלמת פעמים רבות יותר מאחת שמאגדת statements. זו הפוכה מהאינטואיציה של רוב מפתחי ה-OLTP.
- **Retry הוא חלק מהחוזה האפליקטיבי:** אין deadlocks ואין נעילות, אבל יש שגיאות קונפליקט ב-commit. קוד שלא מטפל ב-retry יישבר בפרודקשן דווקא תחת עומס.
- **בחירת המפתחות קובעת את התקרה:** מספרי הדמו התקבלו עם UUIDv4 ופיזור אחיד. hot write key יחזיר את ההתנהגות לעולם שממנו ניסו לברוח.
- **הגבולות התפעוליים דורשים החלטות ארכיטקטוניות מראש:** 5 דקות לטרנזקציה, 3,000 שורות לשינוי, 128MB working memory ושעה לחיבור — כולם מתורגמים לפיצול batch, ל-connection pool בצד הקליינט עם max lifetime של כ-55 דקות, ולהימנעות משאילתות ארוכות.
- **בדקו את ההתאמה לפני שמתחייבים:** השירות מיועד ל-workload OLTP. אנליטי, batch כבד או טרנזקציות ארוכות אינם המקרה שלו — וגם הדוברים עצמם לא הציגו אותו כברירת מחדל חדשה.
- **נקודות התחלה שהוצעו:** ה-playground של DSQL שפועל בדפדפן ללא חשבון AWS וללא setup, דוגמאות ב-GitHub, וה-MCP Server של DSQL [41:39].

# מונחון

| מונח | פירוש כפי שהוצג בסשן |
|---|---|
| Aurora DSQL | בסיס נתונים רלציוני מבוזר ו-serverless, תואם PostgreSQL, עם ACID מלא ו-strong consistency, שיצא ל-GA בשנה שקדמה לסשן. |
| Query Processor (QP) | מיקרו-VM שמריץ PostgreSQL ומהווה את הקונקשן — הרכיב היחיד בארכיטקטורה שמבצע parsing, planning ו-execution. |
| Adjudicator | "השופט": הרכיב שמכריע בזמן ה-commit אם קיימת התנגשות עם טרנזקציה אחרת על אותן שורות. |
| Journal | distributed commit log שאחראי על ה-durability; העותק שלו בריג'ן ה-witness משמש tiebreaker לקוורום. |
| Witness Region | ריג'ן שלישי ללא endpoint, מחזיק Replicated Journal בלבד, ומשמש להכרעה כשאחד הריג'נים נופל. |
| OCC | Optimistic Concurrency Control — אין נעילות במהלך הטרנזקציה; ההתנגשויות נבדקות רק ב-commit, והמנצחת היא הראשונה שביצעה commit. |
| Snapshot Isolation | רמת הבידוד בפועל — חזקה מ-read committed, ברירת המחדל של PostgreSQL. |
| Amazon Time Sync Service | שירות זמן מבוסס שעונים אטומיים וקליטת GPS, שמספק דיוק ברמת ננו-שניות ומחליף את ה-Transaction ID כמנגנון הסדר. |
| DPU | Data Processing Unit — יחידת החיוב שמודדת בתים שנקראו, בתים שנכתבו, compute, וכתיבות חוצות-ריג'ן. |
| Peering | תהליך חיבור שני קלאסטרים נפרדים לקלאסטר multi-region אחד, מונחה צעד-אחר-צעד בקונסול. |
| DSQL MCP Server | שרת MCP שמאפשר לסוכני AI להתחבר ל-Aurora DSQL עם הבנת סכמה ו-skills; מוגדר ב-endpoint, region ודגל קריאה/כתיבה. |