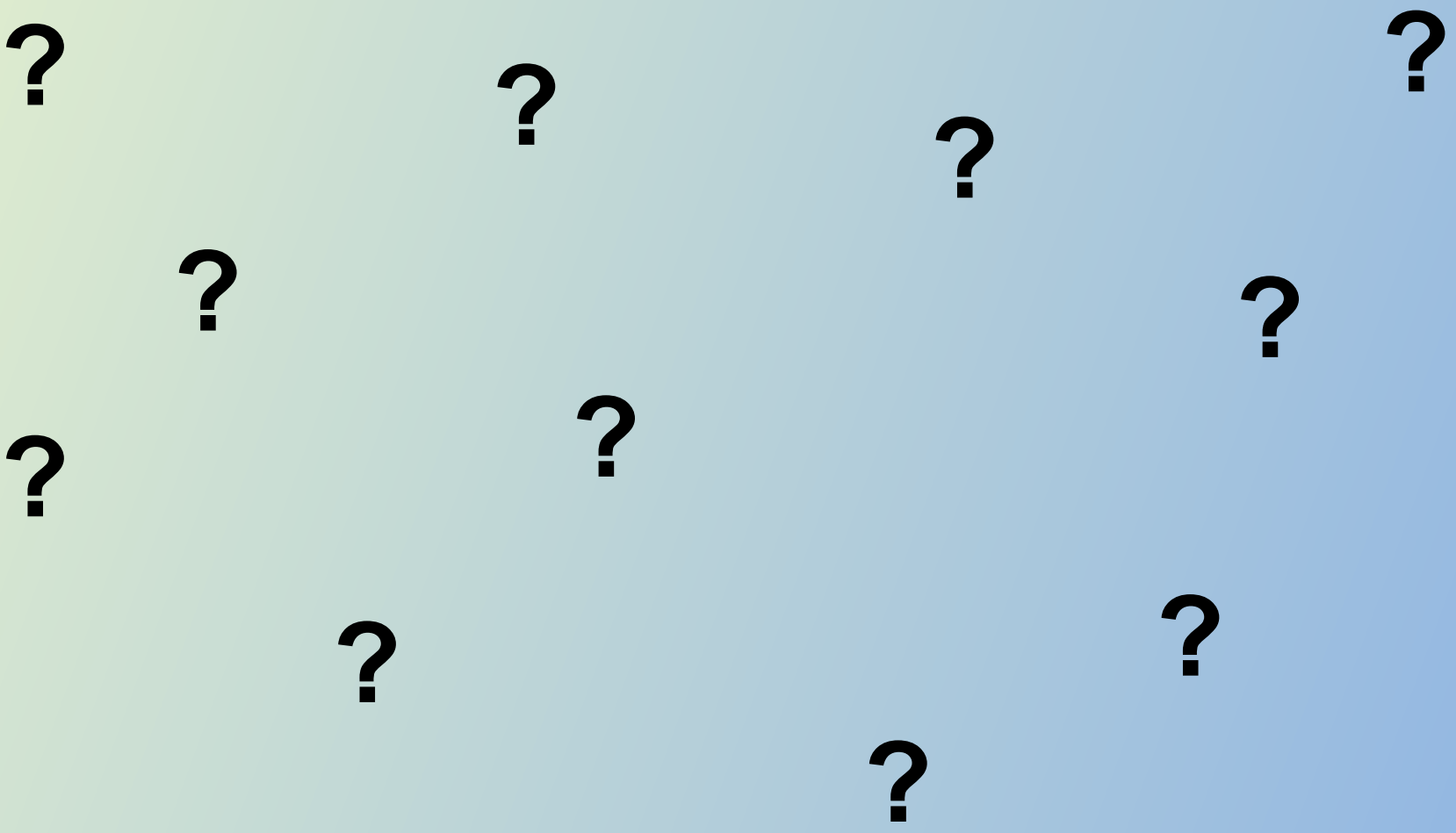
בסיסי נתונים - קורס מתקדם

2023 – תשפ"ג (סמסטר קיץ)

מרצה: רואי זרחיה

שיעור 3

האם יש שאלות משיעור קודם ?



סמכויות והרשאות

אבטחת נתונים בסביבת SQL

בסיס הנתונים מספק **מנגנון להגדרת סמכויות והרשאות** המאפשר להגדיר לכל משתמש מה מותר לו מבחינת:

- גישה לטבלאות הטבלאות
- ביצוע פעולות

בנוסף ניתן להשתמש במנגנון הטבלה המדומה (View) המאפשר:

- לבצע הגנה על שורות / עמודות מסוימות בתוך הטבלה
- לבנות הרשאות על צירופים שונים של טבלאות
- להגדיר לכל משתמש את התצפיות והפעולות המותרות לו

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

מנגנון להגדרת הרשאות

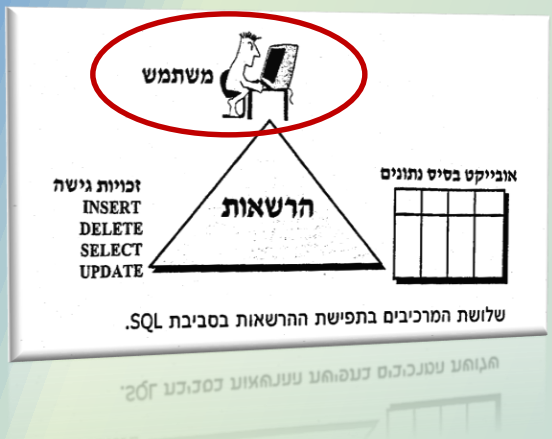
תפיסת ההרשאות בבסיס הנתונים מבוססת על **<u>שלושה מרכיבים</u>**:

- **אובייקט** (Object) עליו מגנים, כגון: טבלה, טבלה מדומה, אינדקס...
- **זכויות** (Privileges) הכוונה לפעולות שמותר למשתמש לבצע על אובייקט כגון: שליפה, עדכון, הוספה ומחיקה
- **משתמש** (User) והזיהוי שלו.
(בהנחה ויש מספר משתמשים עם אותו סוג הרשאה, ניתן להגדיר קבוצה (**Group-ID**) של אנשים שתקבל אותן זכויות גישה (משתמשים באותו ארגון או משתמשים בעלי אותו תפקיד או דרגה).

🡸 לסיכום: **משתמש** מקבל **זכויות** עבור **אובייקטים**

מנגנון להגדרת הרשאות

משולש ההרשאות המתאר: **משתמש** מקבל **זכויות** עבור **אובייקטים**



זיהוי משתמש

מנהל בסיס הנתונים (ה-DBA) הוא זה שמנהל את סמכויות המשתמשים במערכת.

ה-DBA יקצה לכל משתמש בבסיס הנתונים:

- מוקצה שם משתמש חד-ערכי (**User-ID**)
- המשתמש מקבל סיסמא (**Password**)

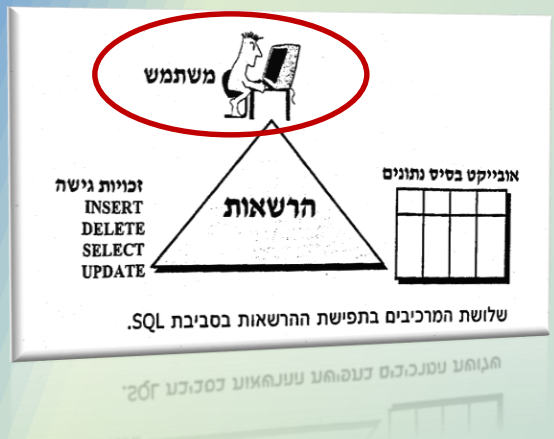
מבנה הפקודה:

CREATE USER *name* **IDENTIFIED BY** 'password';

CREATE USER ‘roei'@‘pink1' **IDENTIFIED BY** ‘123456';

שם המשתמש — שם השרת לכניסה — סיסמת כניסה

זיהוי משתמש



בחלק ממערכות בסיסי הנתונים המסחריות נתוני המשתמש (שם המשתמש וסיסמתו) נלקחים ישירות ממערכת ההפעלה (אותו domain).

מחיקת משתמש:

```
DROP USER ‘roei'@‘pink1';
```

שינוי שם משתמש:

```
RENAME USER ‘roei'@‘pink1' TO ‘omer'@‘pink1';
```

לאחר שהגדרנו את האנשים שיכולים לעבוד על בסיס הנתונים (ע"י מתן סמכויות) נוכל להגדיר את ההרשאות או התפקידים של כל אחד מהם (מה אסור ומה מותר לכל אחד מהם לבצע).

הגדרת זכויות

זכויות הן פעולות שמותר למשתמש מורשה לבצע על אובייקט בבסיס הנתונים.

<u>הזכויות שמשתמש יכול לקבל הן:</u>

- **בחירה** (Select) מתוך טבלה / תצפית (ניתן גם להגדיר את העמודות בטבלה שמותר למשתמש לשלוף מהם)
- **הוספה** (Insert) של שורות חדשות בטבלה / בתצפית
- **עדכון** (Update) של שורות בטבלה / בתצפית (ניתן להגדיר גם את העמודות בטבלה שמותר למשתמש לעדכן)
- **מחיקה** (Delete) של שורות מטבלה / מתצפית
- **ייחוס** (Reference) זכות להתייחס לעמודות של טבלה בבדיקות תקינות או שלמות קשרים כמו במקרה של מפתח זר (לא חייבת להיות הרשאה גישה לטבלה עצמה אלא רק הרשאת ייחוס)

הגדרת אוביקטים

כל אובייקט חדש שנוצר בבסיס הנתונים מכיל מידע על המשתמש שיצר אותו (שם בעל האובייקט).

בעל האובייקט מקבל את כל הזכויות האפשרויות לאובייקט זה ובמסגרת זו יכול בעל האובייקט לבצע את הפעולות הבאות (**פקודות DCL**):

- **להעניק זכויות** גישה לאובייקט למשתמשים אחרים (פקודת **Grant**)
- **להסיר זכויות** שהוענקו למשתמשים אחרים (פקודת **Revoke**)

אם משתמש יוצר טבלה מדומה עליו להיות בעל זכויות גישה לכל טבלאות הבסיס שעליהן בנויה הטבלה המדומה (אך משתמש רגיל יכול לקבל הרשאה לטבלה מדומה גם אם לא רשאי לגשת לטבלאות הבסיס שלה).

הענקת זכויות גישה

פקודת **Grant** מאפשרת להעניק למשתמש זכויות גישה לאוביקטים בבסיס הנתונים (עם אופציה להעניקם גם הלאה למשתמשים אחרים):

מבנה הפקודה:

GRANT privileges **ON** objects **TO** users **[WITH GRANT OPTION]**;

לדוגמא: מתן הרשאת שליפה לדן מתוך טבלת סטודנטים.

GRANT select **ON** Students **TO** Dan;

הענקת זכויות גישה ממוקדות

פקודת Grant מאפשרת גם להעניק למספר משתמשים זכויות מגוונות עם **הגבלות על העמודות** באובייקטים בבסיס הנתונים.

דוגמא: הענקת זכויות שליפה ועדכון לדן ולזאב רק על עמודות מסוימות בטבלת קורסים:

```
GRANT select (courseID, CourseName, Type),
      update (courseID, CourseName, Type)
ON Courses
TO Dan, Zeev
```

הענקת זכויות גישה ממוקדות

<u>דוגמא</u>: הענקת זכות שליפה למספר משתמשים עבור מספר טבלאות עם אפשרות של הענקת זכות הגישה הזו גם הלאה (זאת אומרת למשתמשים נוספים תחתיהם):

```
GRANT select
ON Departments, Students, Courses
TO Dan, Zeev, Avi
WITH GRANT OPTION
```

הענקת זכויות גישה ממוקדות

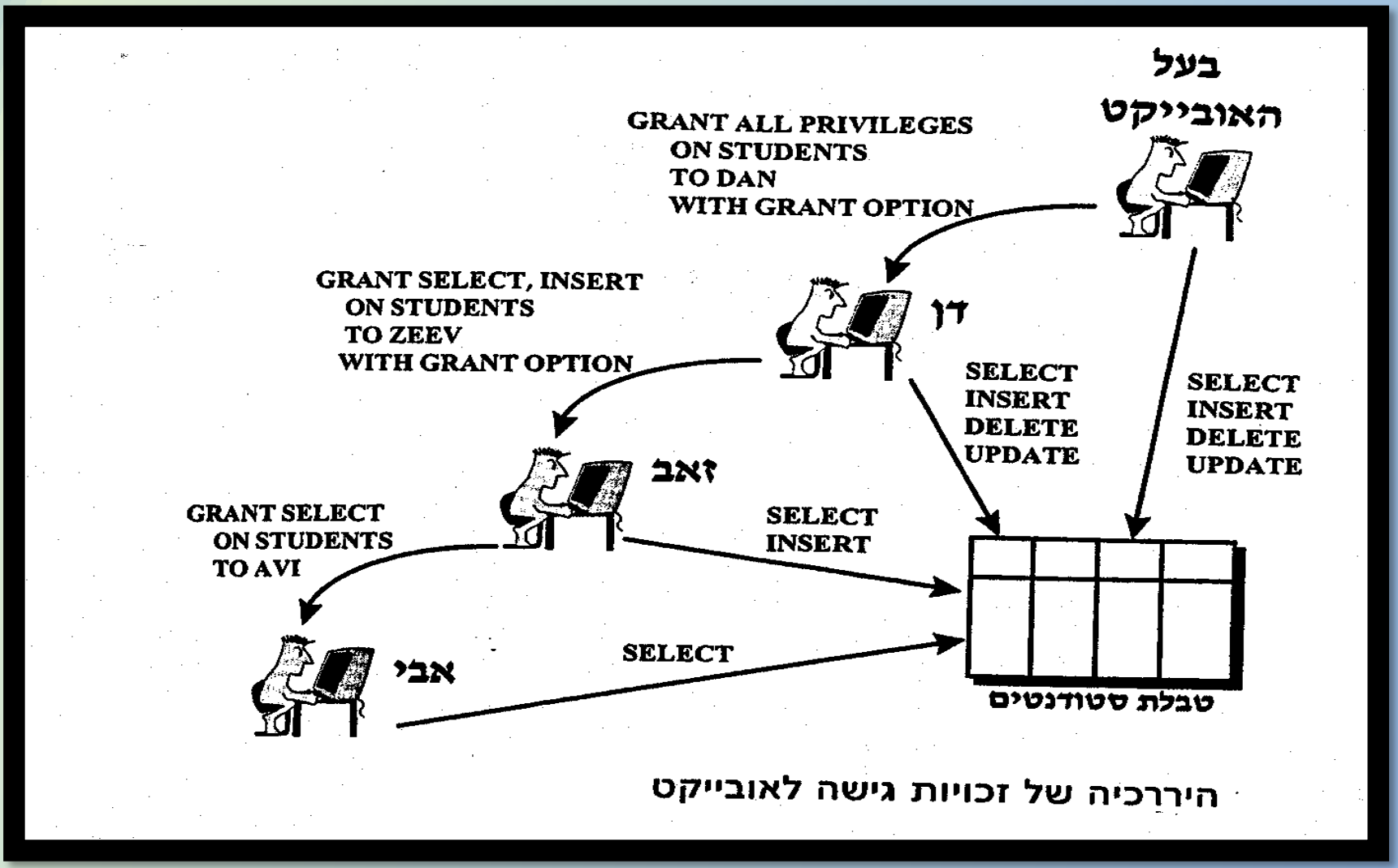
בכדי להגדיר זכויות גישה לטבלה עם **הגבלות על שורות מסוימות** נוכל להגדיר תצפית (VIEW) המכילה תנאי WHERE לסינון השורות הרלוונטיות בלבד ואז מתן זכויות גישה לתצפית.

GRANT select **ON** Haifa_Students **TO** Zeev

כיוון שטבלה מדומה יכולה להיות מבוססת גם על צירוף בין טבלאות שונות אזי נוכל לתת הרשאות גישה למשתמשים דרך טבלה מדומה שמאחוריה מסתתרות מספר טבלאות בסיס.

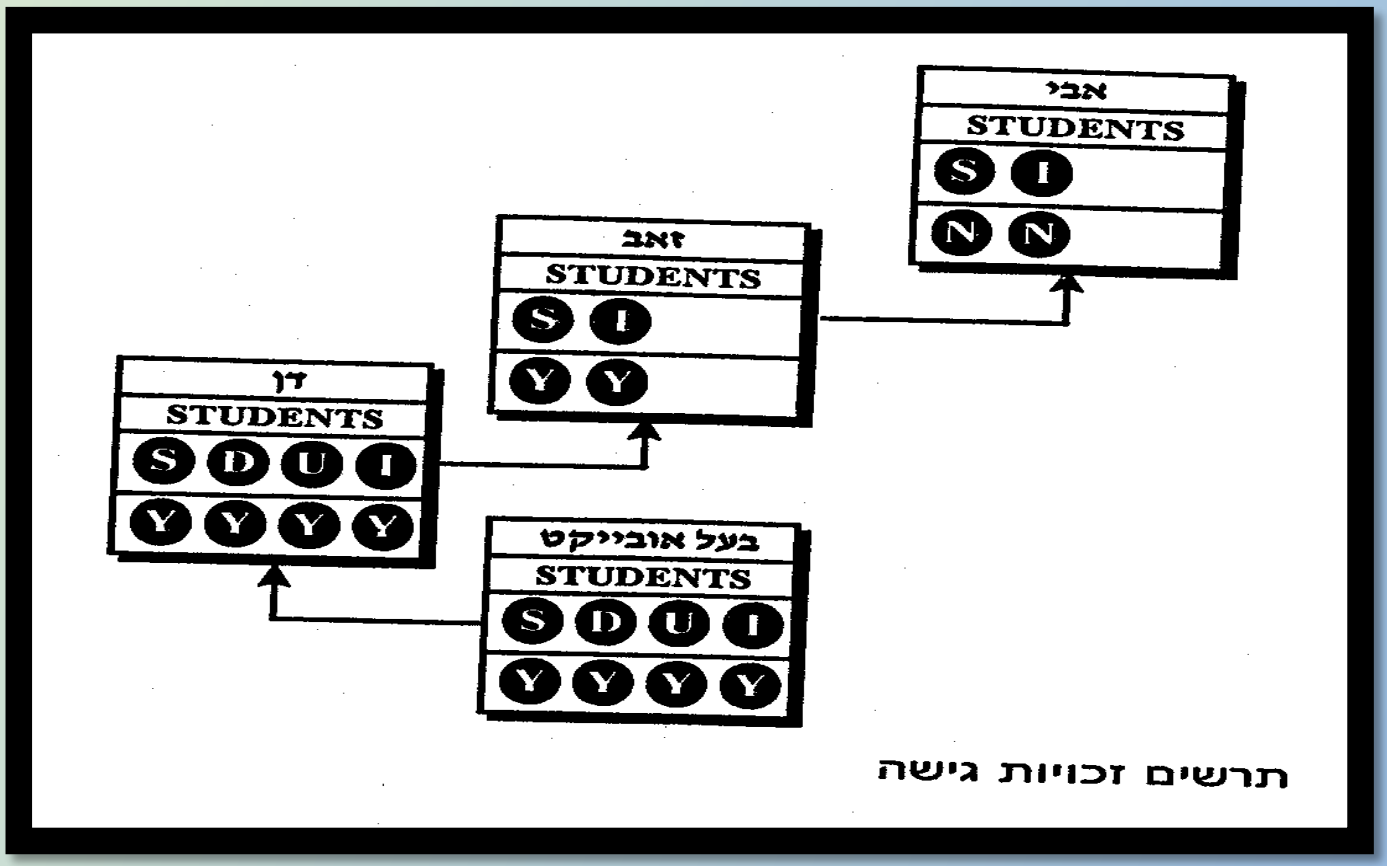
דרך הטבלה המדומה (אם נתנה זכות הוספה) - ניתן לדוגמא להוסיף שורות לטבלאות הבסיס אך זאת במגבלות ההגדרה של הטבלה המדומה (בדיקת תוכן הנתונים שנכנסים לטבלת הבסיס דרך הטבלה המדומה ע"י שימוש באפשרות של **WITH CHECK OPTION**).

שרשור זכויות



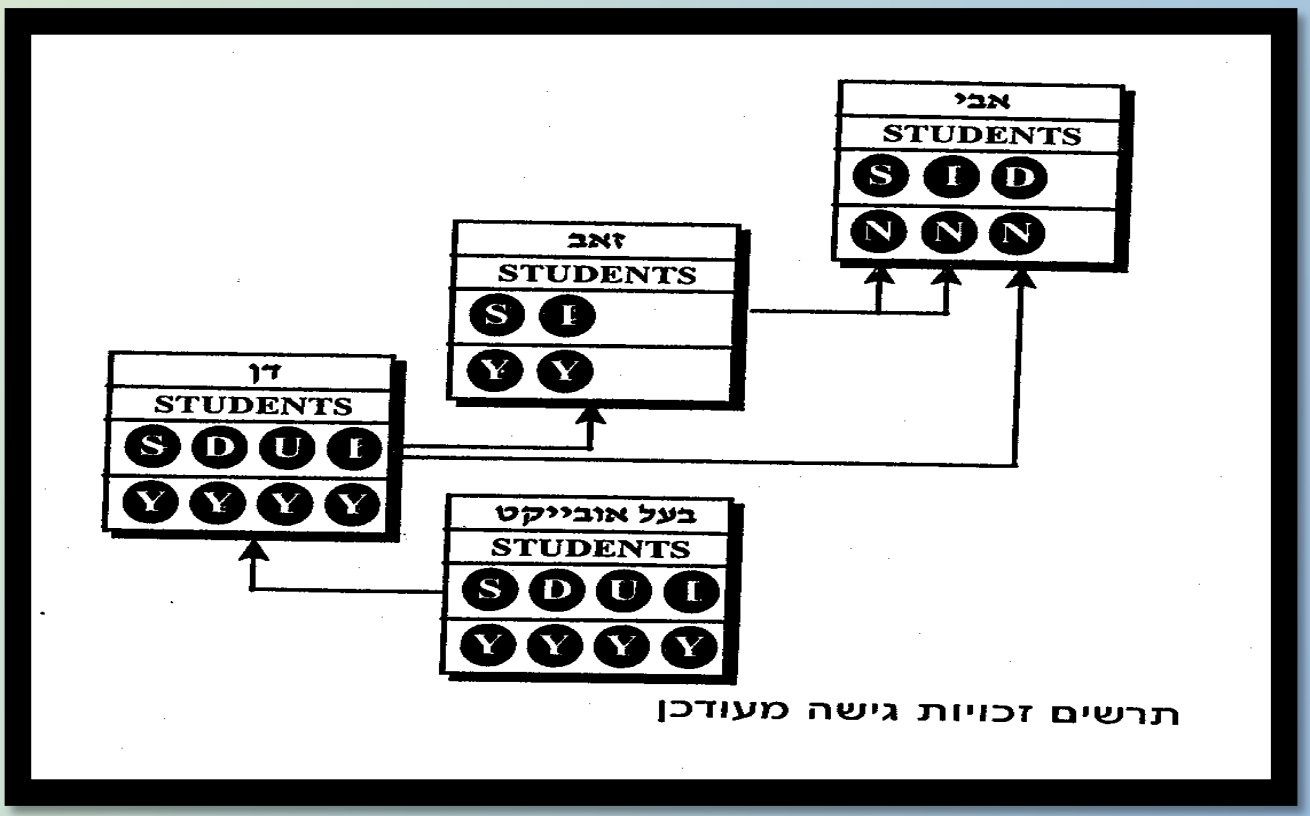
תרשים זכויות

תרשים זה מאפשר למנהל בסיס הנתונים להבין את מצב ההרשאות הנוכחי ומי העניק למי זכויות.



תרשים זכויות - דוגמא

בנוסף להגדרות הקודמות, דן מחליט להעניק זכות מחיקה לאבי:



בעקבות כך, אבי הוא בעל זכויות שליפה והוספה אותן קבל מזאב וזכות מחיקה אותה קבל מדן.

ביטול זכויות גישה

פקודת **Revoke** מאפשרת לבטל את זכות הגישה של משתמש מאובייקטים בבסיס הנתונים (עם אופציה לבטלם גם ממשתמשים אחרים שקיבלו זכויות אלו מהמשתמש הנידון).

<u>מבנה הפקודה:</u>

REVOKE privileges **ON** objects **FROM** users **[** Restrict / Cascade **]**;

הפרמטר CASCADE מבטל את כל היררכית הזכויות שהוענקו ע"י המשתמש שממנו נלקחו זכויות

הפרמטר RESTRICT מונע את ביטול זכות הגישה אם קיימת היררכיה של זכויות (ברירת מחדל)

ביטול זכויות גישה

ביטול כל זכויות הגישה לטבלת סטודנטים למשתמש ששמו דן, כאשר אם דן העניק זכויות גישה למשתמשים נוספים – אזי יש לבטל גם אותן.

<u>מבנה הפקודה:</u>

REVOKE all privileges **ON** students **FROM** Dan **CASCDE**

אם אנחנו לא רוצים לבטל את זכויות הגישה של המשתמשים שקיבלו זכויות מדן, נרשום:

REVOKE all privileges **ON** students **FROM** Dan **RESTRICT**

פעולת ה RESTRICT היא ברירת המחדל כך שלא היינו חייבים לרשום אותה בסוף הפקודה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

ביטול זכויות גישה - דוגמא

ביטול זכות העדכון לזאב עבור טבלת קורסים:

REVOKE update **ON** courses **FROM** Zeev

במצב בו לזאב היו זכויות עדכון נוספות לטבלאות אחרות, הן לא היו נפגעות.

ביטול זכות העדכון של זאב לעמודת type בטבלת קורסים:

REVOKE update(type) **ON** courses **FROM** Zeev

פעולת הביטול הנ"ל לא תפגע ביכולות העדכון של זאב עבור עמודות אחרות בטבלת קורסים.

ריבוי משתמשים

דן וזאב החליטו להעניק זכויות גישה למשתמש אבי.

Dan: **GRANT** select, insert **ON** students **TO** avi
Zeev: **GRANT** select, insert **ON** students **TO** avi

לאחר מכן הוחלט לבטל את זכות הגישה של זאב ע"י הפקודה הבאה:

REVOKE select, insert **ON** students **FROM** zeev **CASCADE**

מפקודה זו אמור להיות מושפע גם אבי שקיבל את ההרשאות הנ"ל מזאב, אך אבי עדיין יהיה בעל זכויות גישה לטבלה שכן הוא קיבל אותה גם מדן.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

הגדרת תפקידים

כמו שאנו מגדירים משתמש ניתן להגדיר אוסף של הרשאות עבור תפקיד מסוים ולא עבור משתמש ספציפי.

הגדרת תפקיד בחברה מסוג "איש מכירות" והקצאת הרשאות ספציפיות עבורו:

```
CREATE ROLE salesManager
GRANT select, insert, delete ON salesTO salesManager
```

מתן הרשאת "איש מכירות" למשתמש משה

```
GRANT salesManager TO moshe
```

ביטול תפקיד:

```
REVOKE ROLE FROM { user | role  }
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טרנזקציות בסביבת SQL

טרנזקציות

טרנזקציה (transaction) הינה רצף פקודות SQL לעדכון בבסיס הנתונים (הנראים למשתמש-הקצה לפעמים כפקודה בודדת אחת).

המונח **טרנזקציה** מתייחס לאוסף הפקודות שמתבצעות **כיחידת עבודה לוגית בודדת**, שיכולה להסתיים **בהצלחה** או **בכישלון** (במקרה של כישלון, כל פעולות העדכון שהתרחשו יבוטלו).

מטרת העל של הטרנזקציה הינה **שמירה על אמינות ושלמות** בסיס הנתונים בסביבה עתירת תנועות ומרובת משתמשים.

ה DBMS צריך להבטיח את הצלחת **רצף פקודות העדכון** ולא רק את הצלחת **הפקודה הבודדת** (בשיטת "הכל או כלום") שכן לא ניתן לבצע טרנזקציה באופן חלקי.

טרנזקציות

דוגמא: הפקדת צ'ק של לקוח מהווה **פעולה בודדת** עבור הלקוח, אך בפועל בסיס הנתונים בבנק מבצע **שתי פעולות עדכון**, שכן העברת כספים מחשבון א' לחשבון ב' מהווה טרנזקציה שמכילה שני עדכונים לביצוע, אחד לכל חשבון.

חשוב להבין שהטרנזקציה הינה תוכנית (המורכבת מפקודה או אוסף פקודות) הניגשת לנתונים ו(אולי) מעדכנת אותם, ז"א שלא כל טרנזקציה מסתיימת בעדכון של בסיס הנתונים.

```
Start Transaction
   SQL STATEMENT  NO. 1;
   SQL STATEMENT  NO. 2;
   SQL STATEMENT  NO. 3;
   SQL STATEMENT  NO. 4;
Commit;
```

מבחינת מבנה, זו היא יחידה המכילה אוסף פעולות על בסיס הנתונים שבתחילתה ובסיומה המצב הרגעי של בסיס הנתונים נקבע באופן חד ערכי.

טרנזקציות

כפי שצויין, כל הפקודות הקיימות בטרנזקציה יתבצעו בשיטת "**הכל או כלום**", ז"א או שיבוצעו כל הפקודות בטרנזקציה, או לחלופין במצב של כישלון לא תבוצענה אף פעולה (והמצב המקורי ישמר).

דוגמא: לא יתכן מצב שבו מבוצעת הפקדת צ'ק כאשר החשבון של הזכאי מזוכה אך החשבון של החייב לא מחויב (העקביות חייבת להישמר).

על מנת לשמור על שלמות הנתונים, נדרוש שבסיס הנתונים יאפשר את עקרונות היסוד הבאים בהקשר של טרנזקציות:

The ACID properties

- **אטומיות (Atomicity)**
- **עקביות (Consistency)**
- **בידוד (Isolation)**
- **עמידות (Durability)**

עקרונות יסוד לגבי טרנזקציות

אטומיות (Atomicity) - הכל או לא כלום, ז"א שקיימות שתי אפשרויות:

- או שכל פעולות הטרנזקציה יצאו לפועל בשלמותם (executed).
- או במקרה של כישלון, החלק שבוצע יבוטל (undo) ואז שום פעולה לא יצאה לפועל (יוגדר בשפה המקצועית כ-גלגול לאחור – rollback).

הנקודה החשובה להבנה היא שכישלון לא יאפשר להשאיר את מסד הנתונים במצב שטרנזקציה בוצעה באופן חלקי בלבד.

עקביות (Consistency) – תנועה חייבת להעביר את בסיס הנתונים **ממצב תקין אחד למצב תקין אחר** אפילו שתוך כדי פעולתה התנועה מפירה זמנית את תקינות בסיס הנתונים (כמו בדוגמא הקודמת).

* שמירה על העקביות בבסיס הנתונים בהכרח תתקיים בהרצת טרנזקציה במצב שאין טרנזקציות אחרות שרצות במקביל אליה (תרחיש כזה בעצם מבודד את הטרנזקציה הנוכחית משאר הטרנזקציות במערכת).

עקרונות יסוד לגבי טרנזקציות

בידוד (Isolation) - במערכת עם מספר טרנזקציות המבוצעות במקביל, אם לא מתבצעת בקרה לעדכונים המבוצעים למידע המשותף לטרנזקציות הרצות, יש פוטנציאל לאי תאימות שיבוצע ע"י עדכון של טרנזקציות אחרות.

מצב זה יכול לגרור שגיאות עדכון למידע המאוחסן ב-DB ולכן המערכת צריכה לספק מכניזם לבידוד טרנזקציות מהשפעתם של טרנזקציות אחרות הרצות במקביל, זאת אומרת **שתנועות חייבות להתבצע באופן בלתי תלוי זו בזו**.

למרות שניתן להריץ טרנזקציות במקביל, מערכת בסיסי הנתונים מבטיחה שעבור כל שתי טרנזקציות Ti ו Tj ישנן שתי אפשרויות עבור Ti:

- או ש Tj הסתיימה לרוץ לפני ש Ti התחילה.
- או ש Tj התחילה לרוץ רק לאחר ש Ti הסתיימה.

בשיטה זו של **בידוד (Isolation)** קיימת שקיפות לגבי טרנזקציות המבוצעות במקביל, טרנזקציה לא מודעת לשאר הטרנזקציות הרצות במקביל אליה.

עקרונות יסוד לגבי טרנזקציות

עמידות (Durability) לאחר שטרנזקציה הסתיימה בהצלחה, **השינויים** שהיא ביצעה למסד הנתונים **ישמרו במסד הנתונים לתמיד** ולא יאבדו בעתיד בשום תרחיש (גם לא במקרה של בעיה מערכתית כלשהי).

המחשה: על מנת להבין טוב יותר את 4 העקרונות היסוד הנ"ל, נחזור למערכת הבנקאית שמכילה מספר חשבונות ונמחיש את הפעולות שמתרחשות בה כאשר אוסף של טרנזקציות ניגשות ומעדכנות חשבונות אלו.

חשוב להבין שכל טרנזקציה הניגשת למידע משתמשת בפעולות הבאות:

Read() – קריאת מידע מבסיס הנתונים לתוך משתנה שקיים בטרנזקציה.

Write() – כתיבת המידע מתוך המשתנה שקיים בטרנזקציה חזרה לתוך בסיס הנתונים.

עקרונות יסוד לגבי טרנזקציות - דוגמא

דוגמא: נניח שקיימת טרנזקציה Ti שמעבירה $50 מחשבון A לחשבון B.

טרנזקציה זו תוגדר כך:

T_i

```
read(A)
A = A - 50
write(A)
read(B)
B = B + 50
write(B)
```

נבדוק כעת את קיום ארבעת עקרונות היסוד שהוגדרו.

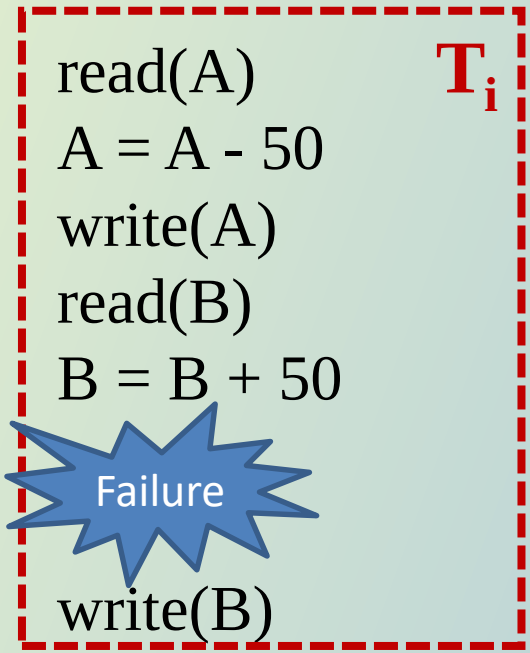
1) עקביות – העקביות הנדרשת כאן היא שסכום הכסף הקיים ב-A וב-B יחדיו לא ישתנה בעקבות הרצת הטרנזקציות, ללא מנגנון העקביות כסף יכול "להיווצר" או "להיעלם" ע"י הרצות שונות (בדוגמא זו מוחסר מ A סכום של $50 ומתווסף ל B סכום של $50 כך שהסכום הכולל נשאר מאוזן).

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

עקרונות יסוד לגבי טרנזקציות - דוגמא

דוגמא: נניח שקיימת טרנזקציה Ti שמעבירה $50 מחשבון A לחשבון B.

2) אטומיות – נניח שלפני הרצת הטרנזקציה Ti , הסכום הקיים בחשבון A היה $1000 והסכום בחשבון B $2000. כעת נניח שבמהלך הרצת הטרנזקציה התרחשה תקלה שמנעה מ Ti להשלים את ההרצה בהצלחה.



במקרה זה הסכום הקיים בחשבון A יהיה $950 (העדכון בוצע) אך הסכום בחשבון B ישאר $2000.

ניתן לראות שהסכום הכולל לפני הטרנזקציה ($3,000) ואחרי הטרנזקציה ($2,950) אינו זהה.

אנו **נרצה להימנע ממצב "לא עיקבי"** מסוג זה (למרות שמתרחש רגעית במסד הנתונים, אך לא נראה לעין) וזו הסיבה לדרישת האטומיות – ז"א, כל הפקודות בטרנזקציה מתרחשות, או שאף אחת לא מתרחשת.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

עקרונות יסוד לגבי טרנזקציות - דוגמא

דוגמא: נניח שקיימת טרנזקציה Ti שמעבירה $50 מחשבון A לחשבון B.

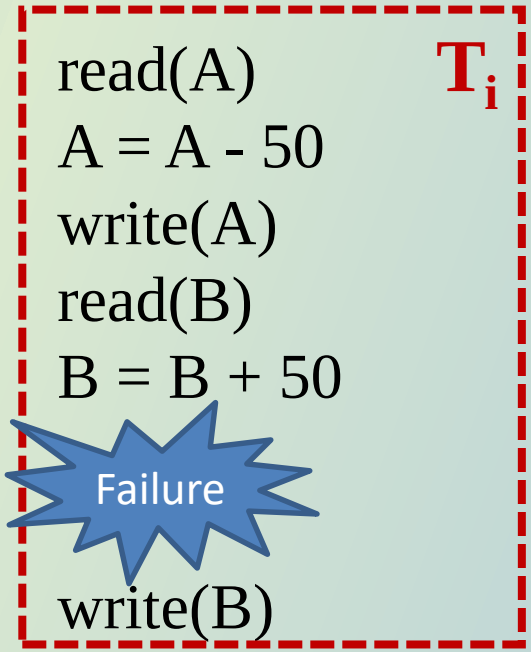
אטומיות – המשך: בפועל, על מנת לשמר את עיקרון האטומיות, בסיס הנתונים עוקב אחר הערכים המקוריים של המשתנים עבורם מתבצעת פעולת Write ואם הטרנזקציה לא מסתיימת בהצלחה (קרי, כישלון), בסיס הנתונים משחזר את הערכים המקוריים כך שבמסד הנתונים תתקבל חזרה התמונה המקורית, ז"א כאילו שהטרנזקציה כלל לא פעלה (ע"י ביצוע גלגול לאחור).

3) עמידות: תכונת העמידות מבטיחה שברגע שטרנזקציה הושלמה בהצלחה ומסד הנתונים עודכן בערכים חדשים, ערכים אלו לא יפגעו או ייעלמו בעקבות כישלונות של טרנזקציות מוצלחות או לא מוצלחות אחרות שיקרו בעתיד.

עקרונות יסוד לגבי טרנזקציות - דוגמא

דוגמא: נניח שקיימת טרנזקציה Ti שמעבירה $50 מחשבון A לחשבון B.

4) בידוד: גם כאשר תכונות האטומיות והעקביות מובטחות לכל טרנזקציה שהיא, במקרה בו מספר טרנזקציות מורצות במקביל, הפעולות שלהן יכולות להתערבב ולגרום למצב של חוסר עקביות.



<u>בדוגמא שלנו</u>: בהנחה ומתרחש **כישלון** כמתואר בטרנזקציה Ti (לפני כתיבת B) **ובאותה שנייה מתבצעת קריאה של משתנים A,B** ע"י טרנזקציה חדשה Tj שמחשבת את A+B אזי נקבל סכום שגוי.

מקרה חמור נוסף יכול להתרחש במקרה ובו בוצעה כבר כתיבה של B ע"י Ti (לאחר הכישלון) ואז Tj תבצע כתיבה של ערכי A,B ואז יתרחש מצב של חוסר עקביות למרות ששתי הטרנזקציות הסתיימו בהצלחה.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טרנזקציות – דוגמא נוספת

קיימות לנו 2 טבלאות (קורסים וציונים) כאשר ידוע שהרישום לקורסים קיימים טרם הסתיים.

כעת ברצוננו לבצע רישום לסטודנט חדש (בעל מס' ת.ז. 210) לקורס קיים שהקוד שלו הוא M-100 (קורס אלגברה לינארית) לסמסטר קיץ 2023.

כעת, לפני שנבצע הוספה של סטודנט נבדוק שיש מקום פנוי בקורס הנוכחי:

Courses

COURSE_ID	COURSE_NAME	TYPE	POINTS	DEPARTMENT_ID	MAX_ENROLL	CURR_ENROLL
מס .קורס	שם קורס	סוג קורס	נק .זכות	קוד מחלקה	מס .נרשמים מקסימלי	מס .נרשמים נוכחי
C-200	Programming	LAB	4	CS	85	38
C-300	Pascal	LAB	4	CS	110	80
C-55	Data Base	CLASS	3	CS	45	10
M-100	Linear Algebra	CLASS	3	MT	90	88
M-200	Numeric Analysis	CLASS	3	MT	125	100
B-10	Marketing	CLASS	2	BS	40	40
B-40	Operations Res.	SEMIN	3	BS	40	36

מבנה טבלת הקורסים לאחר הוספת שתי עמודות.

טרנזקציות – דוגמא נוספת

בהנחה ויש מקום פנוי בקורס נצטרך לבצע רישום של סטודנט בעל ת.ז. 210 לקורס **M-100** לסמסטר קיץ 2023 (הוספת רשומת סטודנט) :

Grades **ציונים**

STUDENT_ID	COURSE_ID	SEMESTER	TERM	GRADE
מס. סטודנט	מס. קורס	סמסטר	מועד	ציון
105	C-55	SUM2007	A	70
210	M-100	SUM2023	A	NULL
105	M-100	SUM2007	B	50
105	C-200	AUT2008	A	85
210	C-200	AUT2008	A	80
210	B-10	WIN2008	A	50
105	B-40	WIN2008	B	70
245	M-100	AUT2008	A	80
245	B-10	AUT2008	A	70
200	C-200	AUT2008	B	50
200	B-10	AUT2008	A	65
245	B-40	WIN2007	A	95
200	M-100	SUM2007	B	90
310	M-100	SUM2007	A	100

טרנזקציות – דוגמא נוספת

לסיכום: התהליך של רישום הסטודנט בפועל יצריך לבצע שתי פעולות:

1) **הוספת שורה חדשה** לטבלת ציונים (עמודת הציון תשאר ריקה):

```
INSERT INTO GRADES (Course_id, Student_id, Semester, Term)
VALUES ('M-100','210','SUM2023','A')
```

2) **עדכון מספר** הסטודנטים שנרשמו לקורס בטבלת קורסים (ביצוע עדכון של מספר הסטודנטים הרשומים להיות 89 במקום 88):

```
UPDATE      COURSES
SET         CUR_ENROLL = CURR_ENROLL+1
WHERE       COURSE_ID = 'M-100'
```


בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טרנזקציות – דוגמא נוספת

<u>לסיכום</u>: ניתן לראות כעת, שעל מנת לבצע את הפעולה שהתבקשנו יש צורך בביצוע רצף של שתי פקודות, כאשר הפקודות חייבות להתבצע כאילו הן פקודה אחת, אחרת בסיס הנתונים יהיה משובש ולא אמין.

<u>דוגמא למקרה של שיבוש בסיס הנתונים:</u>

- הוספת הסטודנט ללא עדכון מספר הסטודנטים בקורס.

הצלחה / כישלון של טרנזקציה

על מנת להתחיל טרנזקציה חדשה נרשום START TRANSACTION ולאחריה יופיע סט של פקודות DML , כאשר סיום טרנזקציה יכול להתבצע או **בכישלון** (זאת אומרת שהופעלה פקודת ROLLBACK) או **בהצלחה** (על ידי פעולת COMMIT).

<u>סיום טרנזקציה בהצלחה</u>

במקרה של הצלחה, כל פקודות ה-DML תחת הטרנזקציה ייבוצעו בפועל והשינויים שבוצעו בעקבותיהם יישמרו בבסיס הנתונים באופן תמידי וסופי.

<u>סיום טרנזקציה בכשלון</u>

מקרה של כישלון, כל פקודות ה-DML תחת הטרנזקציה ייכשלו ויבוצע ביטול של כל הפקודות הנ"ל ע"י הפעלת פקודת ROLLBACK.

אישור סיום טרנזקציה

על מנת שניתן יהיה להודיע ל DBMS שטרנזקציה הסתיימה, יש לשלב בסוף אוסף הפקודות את הפקודה **COMMIT**.

פקודת ה COMMIT מאפשרת לתוכנית היישום להודיע ל DBMS שהטרנזקציה על כל פקודותיה הסתימה בהצלחה וכחלק מכך כל פקודות העדכון שהיו צריכות להתבצע בוצעו ובסיס הנתונים חזר בסיומם להיות במצב תקין (ז"א, שלא היו הפרעות באמצע וש-4 עקרונות ה ACID התקיימו).

אישור סיום טרנזקציה בדוגמא האחרונה

```
START TRANSACTION;

INSERT INTO GRADES (Course_id, Student_id, Semester, Term)
VALUES ('M-100’ , '210’ , 'SUM2023’ , 'A')

UPDATE      COURSES
SET         CUR_ENROLL = CURR_ENROLL + 1
WHERE       COURSE_ID = 'M-100‘

COMMIT
```

פקודת הגלילה לאחור – Roll Back

במקרה של כישלון של אחת הפקודות בטרנזקציה (מסיבה כלשהיא) נצטרך לעצור ולבטל את כל הפקודות שקרו מתחילת הטרנזקציה ועד לרגע זה.

פקודת **הגלילה לאחור** היא זו שמאפשרת לתוכנית היישום לבקש מבסיס הנתונים **לבטל** את כל העדכונים שבוצעו **מתחילת התנועה**.

המנגנון שמאפשר להפעיל את פקודת ה **ROLLBACK** ולחזור למצב המקורי הינו יומן האירועים (**Log File**).

יומן אירועים הינו קובץ המנוהל ע"י ה DBMS ומכיל רשומה עבור כל פקודת עדכון שבוצעה בבסיס הנתונים וכך הוא מאפשר התאוששות ושחזור של המצב המקורי לפי פעולות העדכון.

יומן האירועים

כל רשומה ביומן האירועים תשמור את פרטי המידע הבאים:

- שם התנועה
- הזמן והתאריך בו בוצעה התנועה
- זיהוי המשתמש (שמו וקוד תחנת העבודה שממנו בוצעה התנועה)
- הפעולה שבוצעה (ביטול, עדכון, הוספת, תחילת תנועה, סוף תנועה)
- שם הטבלה שבה בוצעה הפעולה
- תוכן השורה **לפני** העדכון (Before Values)
- תוכן השורה **לאחר** העדכון (After Values)

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

תהליך שיחזור לאחור (Backward Recovery)

זהו תהליך המתרחש ביומן האירועים בעקבות מקרה שבו התעוררה שגיאה שבעקבותיה הופעלה פקודת **ROLLBACK**.

במקרה זה, **יומן האירועים** יקרא בסדר כרונולוגי הפוך, זאת אומרת מהפקודה האחרונה שרשומה לפני פקודת הגלגול האחרון (הפקודה האחרונה שבוצעה) ועד לפקודה הראשונה בתנועה ← המטרה העליונה היא החזרת בסיס הנתונים למצבו שלפני העדכון.

<u>הערה</u>: קיימת גם אפשרות של תהליך שיחזור לפנים (Forward Recovery) של בסיס הנתונים (במקרה של רצון לשחזר את הגיבוי האחרון). במקרה זה, ניתן להפעיל את התהליך על קובץ הגיבוי האחרון שנשמר וכך שתבוצענה כל השורות מיומן האירועים מהפקודה המקורית הראשונה בטרנזקציה וזאת לפי הסדר הכרונולוגי שבו הם בוצעו במקור.

מודל הטרנזקציות (התנועות)

מודל הטרנזקציות מגדיר את האופן שבו ה DBMS מזהה את שלושת השלבים האפשריים בביצוע תנועה בסיסית:

- **תחילת התנועה** - פקודת העדכון הראשונה בתוכנית.
- **סיומה המוצלח של התנועה** - ע"י ביצוע פקודת COMMIT או במקרה בו תוכנית היישום מסתימת.
- **כשלון בעת ביצוע התנועה** - ע"י ביצוע פקודת ROLLBACK או במקרה בו התרחש שגיאה (ERROR) שגרם לקריסת התוכנית

מודל הטרנזקציות – דוגמא להמחשה

<u>לדוגמא</u>: נרצה לבצע טרנזקציה המכילה מספר פקודות DML של שליפה/עדכון/מחיקה אך מסיבה כלשהיא התרחשה שגיאה שהובילה להפעלת פקודת גלגול לאחור וחזרה למצב הבסיס.

לאחר מכן הורצה הטרנזקציה בשנית אך הפעם הסתיימה בהצלחה.

התחלת טרנזקציה START TRAN
- INSERT
- SELECT
- UPDATE

טרנזקציה הסתיימה בכשלון ROLLBACK

התחלת טרנזקציה בשנית START TRAN
- INSERT
- SELECT
- UPDATE

טרנזקציה הסתיימה בהצלחה COMMIT

טרנזקציות – סוגי מצבים

כפי שראינו, טרנזקציה יכולה להסתיים במצב של "הצלחה" ואז העדכונים שביצעה נשמרים בבסיס הנתונים או לחלופין אם התרחש כישלון בזמן ריצת הטרנזקציה (**failure**) אזי הטרנזקציה תעבור למצב מבוטל (**aborted**).

לאור העובדה שקיימת תמיכה בתכונת האטומיות, אזי כישלון של טרנזקציה חייב לגרור את פעולת הגלגול לאחור, משמע לבטל את כל העדכונים שבוצעו משלב הכישלון אחורה ועד לתחילת הטרנזקציה.

ברגע שהעדכונים גולגלו לאחור (**rolled back**) ומצב בסיס הנתונים חזר לקדמותו אזי תכונת האטומיות ("הכל או כלום") אכן נשמרת.

טרנזקציה שהשלימה את ריצתה בהצלחה מסומנת כ **committed**, טרנזקציה כזו שבצעה עדכונים למסד הנתונים, בעצם העבירה אותו ממצב X למצב Y שישמר גם אם יתרחשו כישלונות בעתיד.

טרנזקציות – סוגי מצבים

ברגע שטרנזקציה עברה למצב committed לא נוכל לבטל את העדכונים שביצעה.

הפתרון לביטול עדכונים שבוצעו ע"י טרנזקציה הוא להריץ "טרנזקציית פיצוי" (יוסבר בקצרה בהמשך) שתעשה את הפעולה ההפוכה (לא תמיד ניתן), פעולה זו תתבצע ע"י המשתמש ולא ניתנת לביצוע ע"י מערכת בסיסי הנתונים.

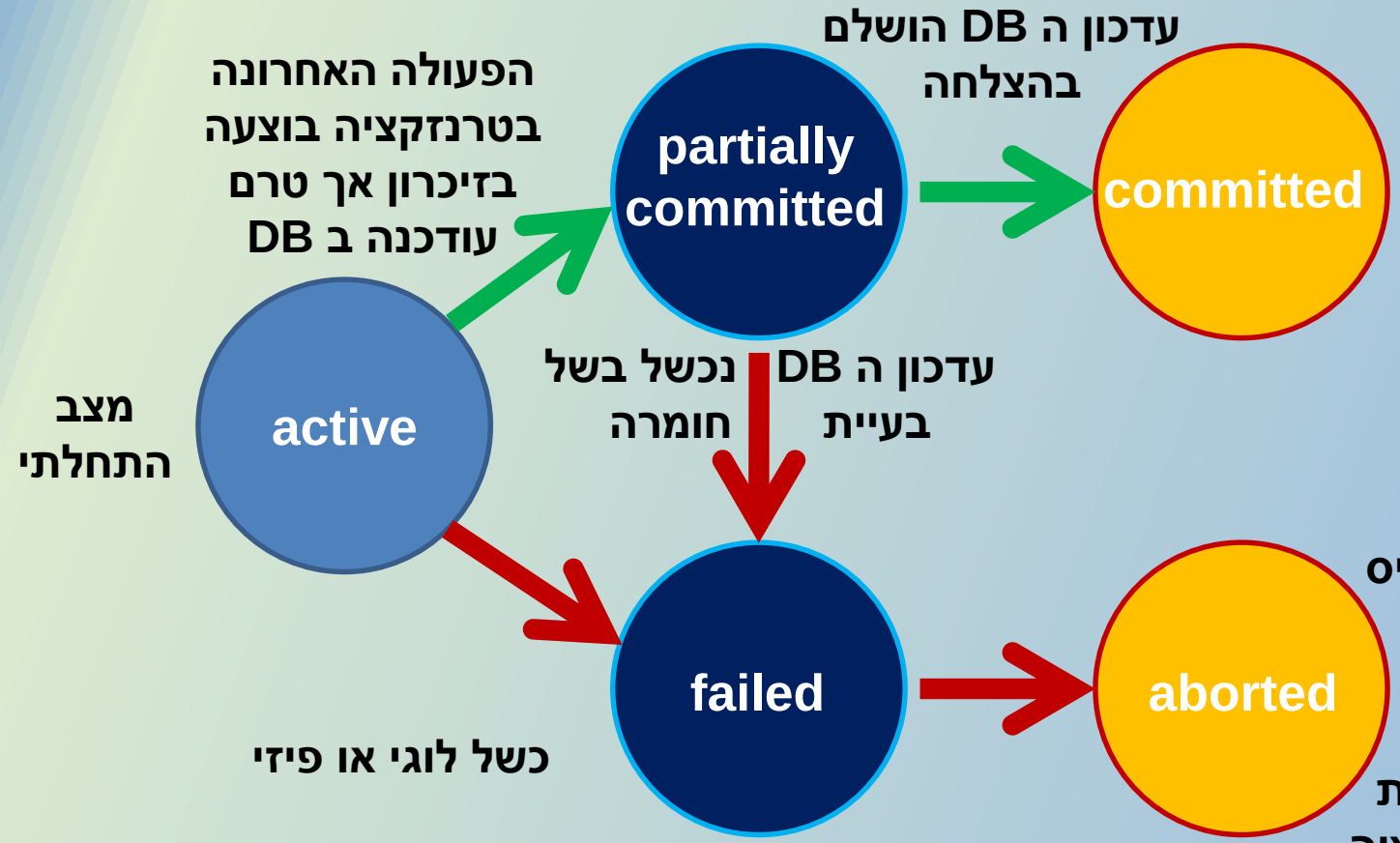
טרנזקציות – סוגי מצבים

על מנת להבהיר בצורה ברורה יותר את הכוונה בהצלחת או כישלון טרנזקציה נגדיר 5 מצבים.

<u>כל טרנזקציה בכל זמן נתון צריכה להיות באחד ממצבים אלו:</u>

- **Active** – מצב ראשוני, הטרנזקציה תישאר במצב זה כל עוד היא רצה.
- **Partially Committed** – לאחר שהפקודה האחרונה בטרנזקציה בוצעה.
- **Failed** – מצב בו לא ניתן לבצע ריצה מלאה ותקינה של הטרנזקציה.
- **Aborted** – לאחר שהטרנזקציה ביצעה גלגול לאחור ובסיס הנתונים שוחזר למצבו המקורי (טרם הריצה).
- **Committed** – מצב של ריצה שהסתיימה בהצלחה.

דיאגראמת מצבים אפשריים לטרנזקציה:



טרנזקציות

<u>הערות</u>:

- טרנזקציה תהיה בסטטוס "הושמדה" (**Terminated**) אם היא הסתיימה במצב של committed או לחלופין במצב של aborted.

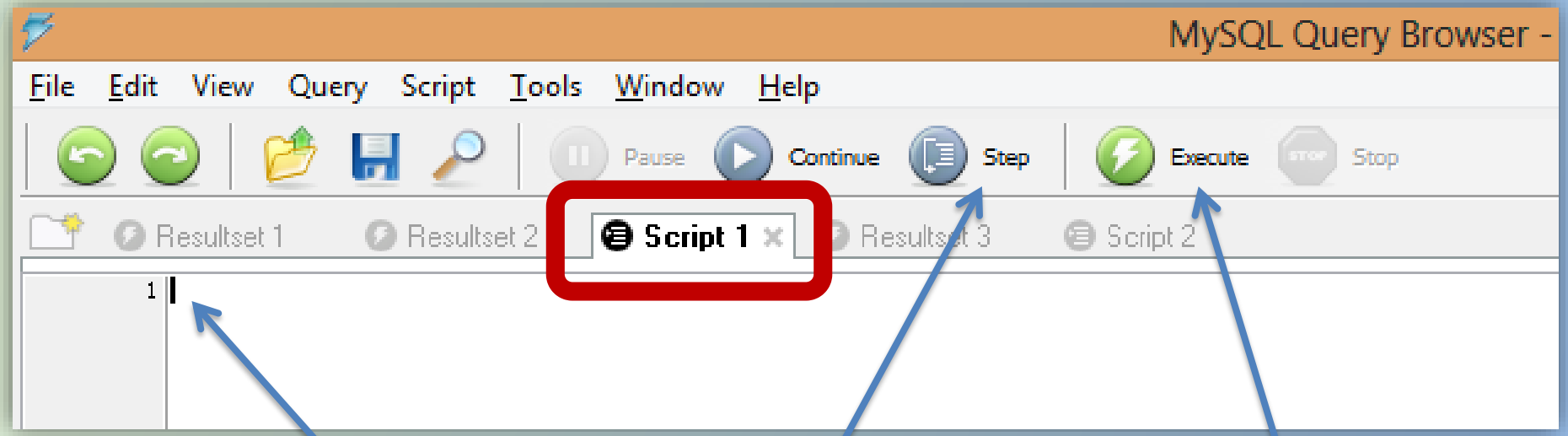
- <u>למה לא נרצה תמיד להמשיך טרנזקציה שבוטלה מהמקום בו הפסקנו ?</u>

ובכן, במקרה של הוצאת כסף מהכספומט לדוגמא, אם הייתה בעיה בזמן שליפת הכסף, לא נרצה לבצע אתחול של המערכת (כ-10 דקות) ורק אז להוציא את הכסף ללקוח, שכן יתכן מאוד שהלקוח כבר לא נמצא ליד הכספומט, ולכן במקרה זה נצטרך להריץ טרנזקציית פיצוי.

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©

טרנזקציות ב MYSQL

לאור העובדה שטרנזקציה הינה בעצם אוסף של פקודות ולא פקודה בודדת אחת (כמו שהיינו מריצים עד כה), אזי נצטרך להשתמש בממשק עבודה חדש בתוך MYSQL הנקרא **SCRIPT** (במקום ResultSet שהשתמשנו עד כה) וזאת ע"י כניסה לתפריט FILE ובחירה באופציה New Script Tab



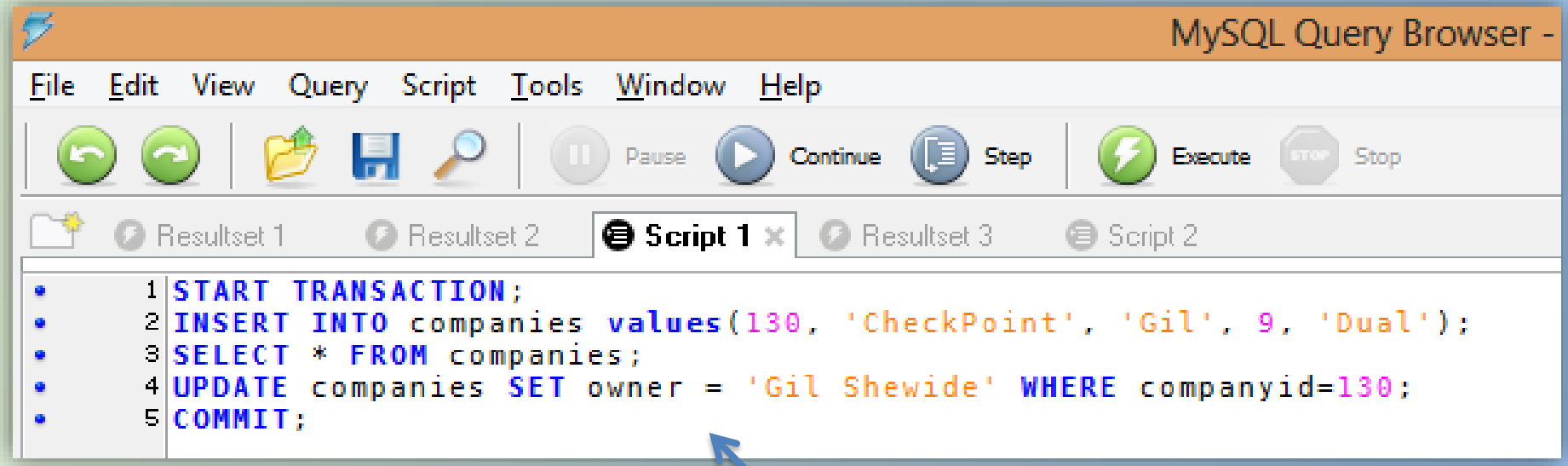
אזור כתיבת הפקודות כשכל פקודה מסתיימת ב " ; "

ביצוע הרצה צעד-אחר-צעד

הרצת טרנזקציה המכילה אוסף פקודות ברצף (סקריפט)

טרנזקציות ב MYSQL

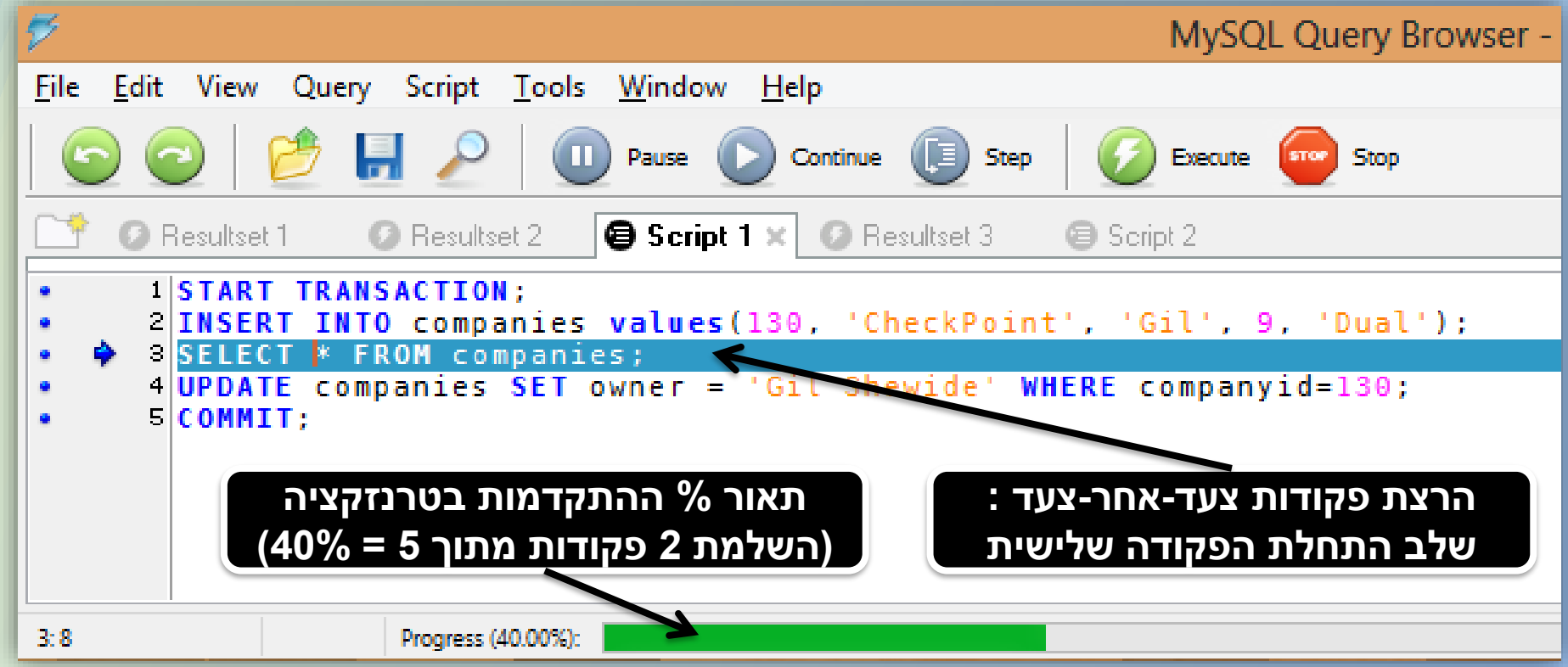
כעת בתוך ה tab של הסקריפט נרשום את הטרנזקציה שלנו המכילה אוסף פקודות DML הרשומות ברצף, אחת אחרי השנייה (סיום פקודה יבוצע ע"י ";" בסוף פקודת ה DML).



פקודת התחלת טרנזקציה המכילה 3 פקודות DML ובסוף פקודת COMMIT

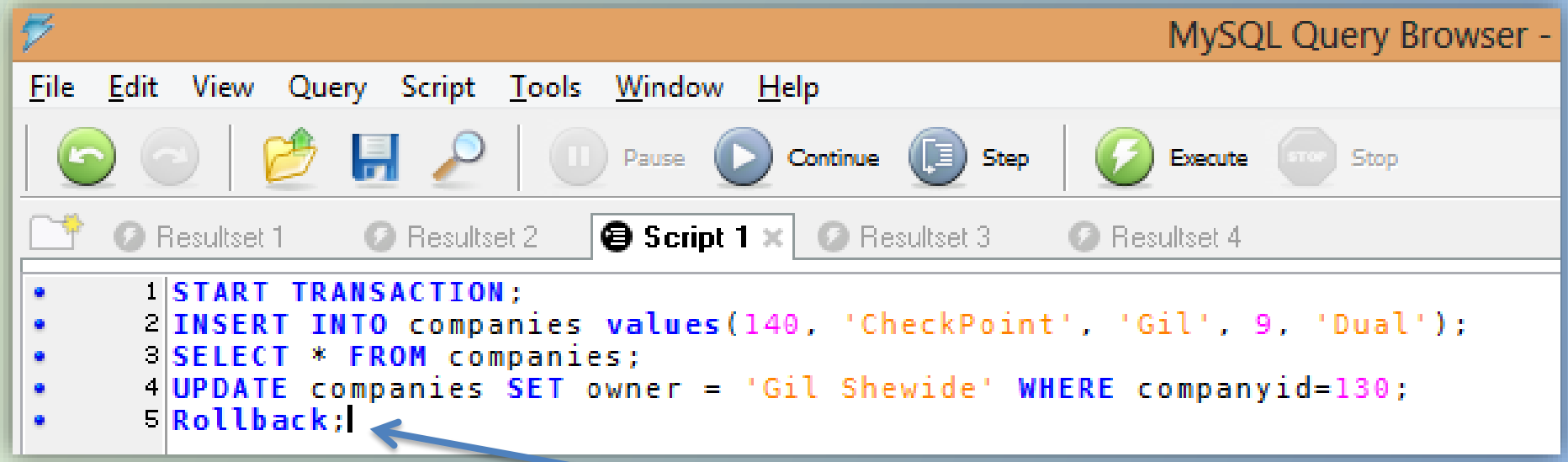
טרנזקציות ב MYSQL

בעת ההרצה, נוכל להריץ את כל הפקודות בפעם אחת ע"י כפתור EXECUTE או לחלופין ע"י כפתור STEP שיאפשר לעבור שורה שורה ולראות את אחוזי ההתקדמות של הטרנזקציה.



טרנזקציות ב MYSQL

במקרה בו היינו כותבים Rollback בסיום הטרנזקציה אזי 3 פקודות ה DML היו מתבצעות (בזיכרון) אך בסיום הייתה מתבצעת פעולת גלגול לאחור שהייתה מחזירה את המצב לקדמותו (טרם התחלת הטרנזקציה) ללא שינוי בבסיס הנתונים עצמו.



פקודת ה ROLLBACK תגלגל לאחור את פקודות ה DML שבוצעו אך טרם נשמרו ב DB

המשך תרגול...

תרגיל כיתה
מס' 2
שאלות 3-7

בסיסי נתונים – קורס מתקדם – רואי זרחיה ©