

# מדריך לעבודה עם Firebird SQL

מאת: עידו קנר

גרסה: 0.2

שינוי אחרון: 02:31:24 07/05/2009



## תוכן עניינים

3	הקדמה
4	היסטוריה
5	כיצד להתחיל לעבוד עם Firebird ?
6	כלים ב Firebird
6	gbak
6	gsec
6	gfix
6	gstat
6	isql
8	להתחיל לעבוד עם isql
8	חלוקה לקבצים
9	עבודה עם קבצי Shadow
11	עבודה עם מסד הנתונים
11	התחברות למסד הנתונים
11	קיצור החיבור
12	עבודה עם טבלאות
14	עבודה עם Blob
16	יצירת כינויים לטיפוסי נתונים
18	אירועים
20	חריגות מותאמות אישית
22	תצוגה
24	הרשאות משתמשים
25	שמירת תוכן של הפלט
26	סיכום
27	תודות

מדריך זה מנסה להסביר כיצד ניתן להתחיל לעבוד עם מסד הנתונים [Firebird](#). המדריך אינו מחליף את התיעוד הרשמי של מסד הנתונים, אלא מהווה נקודת עוגן בשביל להבין טוב יותר את נקודת ההתחלה של עבודה עם מסד הנתונים. החלטתי ליצור מדריך מעט שונה ממה שרובינו מכירים ורגילים בד"כ. המדריך הזה במקום רק להראות כיצד להתחיל, גם מסביר מספר אפשרויות וצורות עבודה אשר יכולות לשפר לכם את חווית השימוש וכתובת התוכנות שלכם עם מסדי נתונים.

מדריך זה הוא בעצם העתק של ה [post](#) שכתבתי בעבר, אך בנוי טיפה שונה, היות ובעיקר הוספתי לו תוכן.

המדריך אינו בא כדי ללמד כיצד לעבוד עם שפת SQL אלא רק איך לעבוד עם דברים ייחודיים או שונים במסד הנתונים הזה אל מול מסדי נתונים אחרים.

במידה שאתם מרגישים כי חסר נושא כזה או אחר, או שצריך להרחיב יותר על נושא מסוים (פרט לכלים המוזכרים במדריך זה), אתם כמובן מוזמנים לבקש ואנסה להוסיף את המידע שאתם זקוקים לו.

המדריך משוחרר תחת הרישיון : [Creative Commons](#) – ייחוס זהה

[Jim Starkey](#) עבד בחברת DEC על מנוע SQL (שעוד לא נקראה אז בשם זה) עבור מסד נתונים מרושת עבור חברת דיגיטל (שנרכשה על ידי HP). צץ ל ג'ים רעיון לפתח מערכת שתוכל לעבוד עם מספר משתמשים בו זמנית בלי לנעול את שאר המשתמשים כאשר מישהו מושך/יוצר/מעדכן מידע (דבר שכיח במסדי נתונים מבוססי יחסי שדות [Foreign keys]). הוא החל לעבוד על הרעיון עבור חברת DEC, אבל מלחמות פנימיות על הפיתוח של Rdb/VMS (אשר כיום מוכר בתור Oracle) גרמו לו לעזוב את החברה.

לאחר עזיבתו את החברה, הגיע לחברת Apollo Computers, אשר חיפשה אנשי מסדי נתונים לפתח מסד נתונים עבור גרסת ה UNIX שלה לתחנות עבודה, והסכימה לממן פיתוח. בעזרת Apollo, ג'ים פיתח את מסד הנתונים Groton Database Systems (על שם המקום בו מסד הנתונים פותח). ב 1984 החלה העבודה על מסד הנתונים, אשר ב 1986 קיבל את השם Interbase.

אף ש Interbase הכניס כסף לחברה, החברה התמודדה עם קשיים כספיים במכירת תוכנה, והחליטה לצאת מהשוק. החברה התחילה ב 1986 להעביר בהדרגה את מסד הנתונים שלה לחברת Ashton-Tate אשר יצרו בין היתר את מסד הנתונים dBASE, אשר רכשה מספר חברות המתעסקות במסדי נתונים. לאט לאט החברה החלה להיכנס לבעיות, וחברת Borland רכשה מהם את Interbase.

בשנת 2000 חברת Borland שחררה לקוד הפתוח גרסה של מסד נתונים [Interbase](#). הגרסה ששחררה הייתה 6.1. הגרסאות הבאות בתור, חזרו להיות קוד סגור. הסיבה לשחרור הקוד של Interbase הייתה בגלל ש Borland רצו חברה נפרדת עבור פיתוח מסד הנתונים, אך בגלל קשיים למצוא חברה שתסכים לתנאים הם שחררו את Interbase בתור קוד פתוח, אבל חזרו מהר מאוד בגרסה 6.5 להחזיק בקוד ללא שחרור התיקונים חזרה.

בעקבות שחרור הקוד של Interbase יצאו כמה פרויקטים אשר המשיכו את הפיתוח של מסד הנתונים בתור קוד פתוח. הפרויקט שקיבל את תשומת לב הרב ביותר היה פרויקט בשם [Firebird](#). לפני מספר שנים, חלק מהפרויקטים הנוספים איחדו כוחות והקוד שלהם נכנס אף הוא ל Firebird, מה שהוליד את גרסה 2 של מסד הנתונים עם תוספות ושינויים, כאשר הבסיס של Borland (כלומר הצורה שמבנה מסד הנתונים עובדה בה) נשאר זהה, ולכן גם מנוע של Interbase מסוגל לעבוד עם קבצי מסד הנתונים של Firebird עד גבול מסוים. שימו לב (נכון לכתיבת המדריך) שעדיין לא כל הפרויקטים אוחדו ל Firebird, וגם גרסה 3 תכיל שילוב קוד נוסף בפרויקט, כדוגמת תמיכה ב SMP ועוד.

עד כאן לחלק ההיסטורי של מסד הנתונים.

## כיצד להתחיל לעבוד עם Firebird ?

נתחיל בכך שנתקין את השרת. החל מגרסה 2, יש 2 אפשרויות לבחירה:

1. שרת קלאסי

2. "סופר" שרת

השרת הקלאסי עובד בצורה המקורית ש Interbase עבד (ואולי אף ממשיך לעבוד) בה. כלומר שרת אחד שפותח תהליכים נוספים על כל בקשה ובקשה, ואף יודע לקבל בקשות של מערכת קבצים, בקשות רשת ועוד. הבעיה עם השרת הקלאסי המקורי הייתה שאם תהליך האב היה קורס, אז היו נשארים לנו התהליכים האחרים, רק בלי תפקוד תקין (מפני שהתהליך הראשון התנהל עם IPC מול שאר התהליכים) כך היה בגרסה 6 - אינני יודע אם זה השתנה בגרסה 7 של Interbase.

הסופר שרת עובד בגישה שונה, בה אנחנו מקבלים תהליך אחד שאחראי על הכל ובכך בעצם השרת מתנהל בתור שרת חוטים (threads) ולא שרת תהליכים.

לא עשיתי בדיקות ביצועים על הסופר שרת, ולכן אני לא יודע עד כמה הצורה שהוא עובד בה טובה או מהירה אל מול השרת הקלאסי.

לאחר שהותקן המנוע (מומלץ להתקין את הסופר שרת), יש להתקין גם את הלקוח (בשביל גישה טובה יותר למסד הנתונים מן הסתם). בלינוקס הכלי הגרפי הכי טוב שאני מכיר לעבודה עם Firebird הוא [Flamerobin](#).

לאחר שנריץ את Firebird (שימו לב שיש הפצות לינוקס שישתמשו ב xinitd בכדי להריץ את השרת, בעוד שבהפצות אחרות הוא יהיה daemon לכל דבר ועניין. הסופר שרת בכל מקרה מכיל שרת בפני עצמו, כך שצריך לוודא שיש תסריט העלאה והורדה של השרת), ניתן לעבוד עם אחדים מהכלים של Firebird.

רשימה קצרה של חלק מהכלים המגיעים עם Firebird אשר מוזכרים כאן במדריך. חשוב לדעת כי אלו לא כל הכלים המגיעים עם Firebird וצריך מדריך נפרד בשביל ללמד כיצד לעבוד עם כל כלי וכלי.

### gbak

הכלי הכי חשוב בעבודה עם Firebird הוא [gbak](#). תפקיד הכלי הוא לבצע כמה דברים ממש חשובים:

1. לגבות את מסד הנתונים (אחת מהדרכים לגבות את המידע, החל מ Firebird 2.0)
2. שדרוג מגרסאות מגרסה ישנה לגרסה חדשה יותר של מסד הנתונים
3. יצירת קובץ מסד נתונים מקובץ גיבוי
4. לשנות תכונות במסד הנתונים (כדוגמת דחיסת המידע, הסרת רשומות מחוקות, השמת ה index לערך הבא הכי קרוב שניתן להשתמש בו ועוד)

ל `gbak` יש עוד דברים שהוא יעזור לנו איתם, אבל אצור מדריך מיוחד בשביל ללמד איך לעבוד עם הכלי.

### gsec

כלי חשוב נוסף הוא `gsec`. תפקיד הכלי הוא ניהול האבטחה של השרת, כאשר הניהול העיקרי הוא של משתמשים. בגרסה 2 של Firebird הקובץ בו נשמרים הנתונים הוא `security2.fdb`. בגרסאות ישנות יותר היו לו שמות אחרים.

כאשר מוסיפים משתמשים, ההוספה עצמה אינה מספקת להם שום יכולות אמיתיות, אלא יש הגדרה פיזית של משתמשים בשייכולו להתחבר למנוע שרת הנתונים. בשביל לתת הרשאות, יש לעשות את זה לפי מסד הנתונים הספציפי בו אנו עובדים, וזה נעשה על מסד הנתונים ללא השימוש ב `gsec`, אך אם המשתמש אינו הוגדר דרך `gsec`, אותו משתמש לא יוכל להתחבר בכלל למסד הנתונים איתו מתבצעת העבודה.

כלומר יש הפרדה בין הגדרת משתמשים לבין הרשאות, וההפרדה הזו מתבצעת לפי מסד נתונים ולא לפי הגדרת משתמש. גישה זו שונה מאוד מכמה מסדי נתונים אחרים הקיימים בעולם.

הסבר מפורט כיצד להשתמש בפקודה תתבצע במדריך נפרד.

### gfix

הכלי `gfix` מאפשר לנהל את מסד הנתונים על ידי דחיסת נתונים, תיקון מידע, מחיקת רשומות שסומנו למחיקה, שינוי תכונות של מסד הנתונים, עבודה עם קבצי `shadow` ועוד.

### gstat

הכלי מספק סטטיסטיקה אודות מסד הנתונים, כדוגמת כמות טבלאות, כמות קבצי `shadow`, כמות חיבורים, מידע על `transaction`, מידע נוסף על מסד נתונים.

## isql

אם המערכת שלכם עובדת עם ODBC בלינוקס/יוניקס, בטח יש לכם עוד כלי בשם זה, שניהם יהיו ממוקמים במקומות שונים. אצלי במנדריבה isql של Firebird ממוקם ב 32 ביט:

```
/usr/lib/firebird/bin/
```

ב 64 ביט:

```
/usr/lib64/firebird/bin/
```

בהפצות שונות המיקום יהיה שונה. למשל ב Arch Linux הפקודה תהיה תחת

```
/opt/firebird/bin/
```

אך השם הוא fb\_isql במקום isql. כך שחשוב מאוד לדעת ולהבין כיצד ההפצה שלכם מתנהגת עם Firebird.

ב gentoo נמצא את שם הפקודה fbsql.

דביאן (וכנראה גם אובונטו) ישימו את הפקודה במיקום הבא:

```
/usr/lib/firebird/bin/isql
```

תפקידו של isql הוא לתת לנו גישה למסד הנתונים שברצוננו לעבוד איתו בתור cli ממש כמו הפקודה mysql של מסד הנתונים MySQL, או psql של PostgreSQL. במדריך זה אדבר על isql וכל העבודה שלנו תתבצע דרכו אלא אם צוין אחרת.

## להתחיל לעבוד עם isql

לפני שאתחיל חשוב לדעת שבניגוד להרבה מסדי נתונים רבים אחרים, Firebird הוא לא case sensitive, כלומר אם יש לנו טבלה בשם Views אז לא משנה אם תבצע הפקודה

```
SELECT * FROM ViEwS;
```

עדיין תתקבל אותה תוצאה כמו ההרצה של:

```
SELECT * FROM Views
```

כאשר הפקודה isql תרוץ בצורה "ערומה", תתקבל הודעה שיש צורך להתחבר או ליצור מסד נתונים.

ע"מ ליצור מסד נתונים בעזרת הסופר שרת (איתו אני עובד בזמן כתיבת מדריך זה) נשתמש בפקודה:

```
SQL>CREATE DATABASE '127.0.0.1:/tmp/firebird.fdb' PAGE_SIZE 8192↵
CON>user 'SYSDBA' password 'masterkey'↵
CON>DEFAULT CHARACTER SET UTF8;↵
```

הוגדרה כתובת השרת (במקרה הזה 127.0.0.1), אחריו הוגדר המיקום בו תהיה אחסנה של מסד הנתונים. יש גם הגדרה של גודל הדפים בהם ישמרו הנתונים ובסוף הוגדרו משתמש ברירת המחדל שיהיה למסד הנתונים וכמובן את הקידוד ברירת המחדל שישמר במסד הנתונים (חשוב לדעת כי כל שדה בטבלה יכול לקבל קידוד אחר מברירת המחדל שהגדרנו כאן).

יש לשים לב כי בברירת מחדל, מקובל להשתמש במשתמש SYSDBA ובססמה של masterkey ב Firebird ו Interbase, אבל בפועל מומלץ כמובן להחליף את זה ולא להשתמש בהגדרות ברירת מחדל לפחות לא לססמה.

חשוב לדעת שברירת המחדל בהגדרות המשתמשים הוא המשתמש SYSDBA אשר יהיה המשתמש היחיד שמוגדר ל Firebird, אך כאמור ניתן לשנות את זה על ידי שימוש ב [gsec](#).

עוד דבר שחשוב לשים לב אליו הוא הגרשיים. כל מקום בו רשמתי גרשיים, הוא מקום חובה (כלומר בנתיב בו ניצור את מסד הנתונים, בשם מהשתמש ובססמה). במידה ולא יהיו גרשיים באותן נקודות, isql ידפיס לנו הודעת שגיאה.

לאחר היצירה יש צורך לכתוב את הפקודה commit עם נקודה פסיק על מנת שכל השינויים ישמרו במסד הנתונים. יש לשים לב שאחרי שנוצר מסד הנתונים, isql כבר מצביע עליו, ואין צורך להתחבר אליו.

### חלוקה לקבצים

כאשר אנחנו יוצרים את מסד הנתונים, בברירת מחדל כל המידע יאוחסן בקובץ אחד. הבעיה מתחילה כאשר יש הרבה מידע, ואז הקובץ שמכיל את המידע נהיה גדול ומנופח. Firebird מספק יכולת להתמודד עם הבעיה הזו, ע"י חלוקה למספר קבצים, בהתאם לגודל הדף שהוגדר למסד הנתונים, וכמות הדפים שברצוננו להחזיק בקובץ אחד במסד הנתונים. כלומר אפשר להגיד שמכמות דפים מסוימת, Firebird צריך לשפוך את שאר המידע לקובץ אחר, ובכך לבזר את המידע למספר קבצים שמרכיבים ביחד את מסד הנתונים.

ברירת המחדל של Firebird לגודל הקובץ (במידה ונבחר באפשרות להשתמש בעוד קבצים), הוא 1024 דפים של מידע, אך ניתן לשנות את זה לערך אחר.

בשביל להגיד ל Firebird להוסיף לנו עוד קובץ עבור מסד הנתונים שלנו, נשתמש בפקודה הבאה:



```
SQL>ALTER DATABASE ADD FILE 'firebird2.fdb';
```

במידה שיש צורך לשנות את גודל הדפים בקובץ החדש, יש להשתמש ב:LENGTH

```
SQL>ALTER DATABASE ADD FILE 'firebird2.fdb' LENGTH 2048;
```

ועכשיו יהיו 2048 דפים בגודל שהגדר למסד הנתונים לפני שיהיה צורך לבזר שוב את המידע. כאשר יוצרים את מסד הנתונים, ויש צורך לשנות את כמות הדפים שתהיה למסד הנתונים, יצרת מסד הנתונים תתרחש בצורה הבאה:

```
SQL>CREATE DATABASE '127.0.0.1:/tmp/firebird.fdb' PAGE_SIZE 8192 LENGTH 2048;
CON>user 'SYSDBA' password 'masterkey';
CON>DEFAULT CHARACTER SET UTF8;
```

הוספת ה LENGTH גם ביצירת מסד הנתונים קובעת את כמות הדפים שיהיו.

## עבודה עם קבצי Shadow

קבצי Shadow הם קבצים המכילים עותקי מידע של מסד הנתונים ובכך מאפשרים יכולת לגבות מידע על גבי מספר מקומות פיזיים בהם הם יאוחסנו, ובכך להבטיח שרידות מידע בצורה טובה יותר. גם כאשר מסד הנתונים עצמו אינו זמין, אך במידה שיש גישה לקובץ shadow אחד לפחות (ניתן ליצור יותר מקובץ shadow אחד עבור כל מסד נתונים), הנתונים של מסד הנתונים עדיין יהיו זמינים, ואף ניתן לקחת אותם וליצור מהם את מסד הנתונים מחדש ע"י שימוש ב [gbak](#).

ניתן ליצור קבצי shadow לפי כמות דפים שבשימוש, כאשר יהיה סוג של טווחים מכמות דפים ועד כמות דפים שבשימוש.

ניתן על ידי אסטרטגיית ניהול להחליט שלא כל קבצי ה shadow שנוצרו יהיו פעילים או בשימוש, וניתן להפעיל, להסיר, ליצור ולמחוק קבצי shadow בכל זמן תחזוקה נתון.

בשביל ליצור קבצי shadow נשתמש בתחביר הבא:

```
SQL>CREATE SHADOW 1 AUTO '/home/shadow1/firebird.shd1';
SQL>CREATE SHADOW 2 MANUAL '/home/shadow2/firebird.shd2';
```

הפקודות למעלה אומרות למסד הנתונים ליצור 2 קבצי Shadow במיקומים שונים, כאשר אחד מנוהל בצורה אוטומטית. כלומר מה קורה כאשר קובץ ה shadow לא נמצא, מה קורה כאשר הוא כן קיים (הצורה בהתנהגות של החיבור אל הקובץ) ואיך להתנהג כאשר מוחקים את קובץ ה shadow.

הניהול הידני אומר ל Firebird להיכשל בכל פעולה כל עוד קובץ ה shadow לא מחובר למסד הנתונים.

במידה וקובץ ה Shadow מוגדר בצורה הבאה:

```
SQL>CREATE SHADOW 3 CONDITIONAL '/home/shadow3/firebird.shd3';
```

אז גם אם החיבור אל הקובץ shadow נכשל, השימוש במסד הנתונים ימשיך כרגיל. הפעלה וניתוק של קבצי Shadow יתבצעו על ידי הפקודה [gfix](#) אשר צריך לציין לה בדיוק מה לחבר ולנתק בצורה הבאה:

```
# gfix -activate /home/shadow1/firebird.shd1
```

הפקודה תפעיל את קובץ ה shadow.

במידה ונרצה להסיר את כל קבצי ה shadow הלא פעילים, נעשה את זה בצורה הבאה:

```
# gfix -kill /tmp/firebird.fdb
```

הסיבה להסרת קבצי shadow לא פעילים, היא מפני שמסד הנתונים יפסיק לתפקד כמו שצריך עד אשר הם יהפכו להיות פעילים או שהם יוסרו לגמרי ממסד הנתונים, ובכך הוא יוכל להמשיך ללא בעיות.

ניתן גם להשתמש בפקודה DROP SHADOW בכדי להסיר קובץ shadow מסוים, לשם כך יש לבדוק בתיעוד של Firebird.

על מנת לבדוק האם מוגדר למסד הנתונים שלנו קבצי Shadow אפשר לעשות 2 פעולות:

1. עבודה עם [gstat](#)

2. הרצת הפקודה show database.

עבודה עם [gstat](#), בשורת הפקודה מתבצעת בצורה הבאה:

```
# gstat -header firebird | grep -i shadow
```

הפקודה תחזיר את הפלט הבא:

```
Shadow count 3
```

הסיבה שהתקבלו 3 קבצי shadow היא מכיוון שהוגדרו על ידנו 3 סוגי חיבורים ל Shadow בהדגמה למעלה.

דרך נוספת לקבל מידע אודות טבלאות ה shadow היא השימוש ב show database:

```
SQL>show database;
```

הפלט יהיה:

```
Database: firebird
Owner: SYSDBA
Shadow 1: "/home/shadow1/firebird.shd1" auto
Shadow 2: "/home/shadow2/firebird.shd2" manual
Shadow 3: "/home/shadow3/firebird.shd3" conditional
...
```

ושאר הפלט ש show database מספק.

### התחברות למסד הנתונים

בשביל להתחבר למסד נתונים יש צורך להשתמש בפקודה הבאה:

```
SQL>CONNECT 127.0.0.1:/tmp/firebird.fdb user 'SYSDBA' password 'masterkey';
```

במידה ומסד הנתונים באמת קיים, אתם תתחברו אליו. יש לשים לב שאם לא צוינה הססמה אז תתקבל הודעת שגיאה על המשתמש שנעשה בו שימוש.

חשוב לדעת של Firebird יש תמיכה בכמה דיאלקטים של SQL. כל דיאלקט מוסיף תמיכה בעוד דברים כדוגמת תאריך ושעה, מערכים וכו'. בד"כ נהוג לעבוד עם דיאלקט מספר 3 אשר מכיל את כל התכונות שיש ל Firebird להציע.

### קיצור החיבור

ניתן לקצר את פעולת החיבור עלי ידי שימוש ב alias. לשם הגדרת ה alias יש לגשת לקובץ aliases.conf ולהוסיף שורה של מסד הנתונים בצורה הבאה:

```
fb_db = /tmp/firebird.fdb
```

לאחר מכן בחיבור עם isql אפשר להתחבר בצורה הבאה:

```
SQL>CONNECT "localhost:fb_db" user 'sysdba' password 'masterkey';
```

הכינוי שנוצר (alias) אומר לשרת איפה למצוא את מסד הנתונים. השימוש בכינוי מגדיר שהחיבור יהיה מעל TCP (דוגמה למעלה החיבור הוא ב localhost).

לאחר החיבור למסד הנתונים, אפשר להריץ את הפקודה

```
SQL>SHOW TABLES;↵
```

היות ומסד הנתונים רק נוצר, תתקבל כמובן הודעה שאין שום טבלאות במסד הנתונים. העניין הוא שיש מעצם היצירה כבר טבלאות במד הנתונים.

ב Firebird יש טבלאות פנימיות אשר שומרות מידע רב ומגדירות את דרך העבודה של מסד הנתונים. כל שם שמור (כלומר פנימי) של טבלה או של שדה ב Firebird יתחיל עם RDB\$ (יש לשים לב שהדולר נמצא בסוף). בשביל לראות את הטבלאות האלו, יש להשתמש בפקודה

```
SQL>SHOW SYSTEM;
```

לאחר הרצת הפקודה תתקבל רשימה של כל הטבלאות הפנימיות (מערכת) שקיימות למסד הנתונים. לצפייה בתוכן של הטבלאות האלו, אפשר להשתמש בפקודת select הרגילה.

למשל:

```
SQL>SELECT * FROM RDB$PAGES;↵
```

אחת הטבלאות הפנימיות היא טבלה המכילה את התיאור שניתן לספק לכל שדה שנוצר עם create table, כלומר תיעוד של שדה בטבלה שאנחנו יורצים, כמובן שאפשר ליצור גם תעוד לטבלאות עצמם, תיאור stored procedure, trigger ועוד ...

**אזהרה: אין לשנות ידנית ערכים אלו ללא הבנת המשמעות! שינוי כלשהו בהם עלול להרוס את מסד הנתונים לגמרי**

כאשר יוגדר טבלה עם אינדקס אשר מעלה את הערך בכל הכנסה של מידע (כלומר Auto Increment) יהיה צורך ליצור Generator או בשמו היותר [RDBMS Sequence](#) (הפקודה/מונח של sequence נכנסה לגרסה 2 של Firebird, ועד אז הייתה ידועה בתור Generator בעבודה עם Firebird).

הרבה אנשים מרימים גבה כאשר מדברים על sequence בגלל שהדבר נראה להם overkill אל מול מסד נתונים כדוגמת MySQL, אבל כאשר מתחילות בעיות [כאלו](#), מבינים כמה חשוב ערך שאנחנו יודעים שישמר בלי קשר למצב הנתונים בטבלה. שימו לב שגם PostgreSQL ומרבית מסדי הנתונים בעולם משתמשים ב sequence בדיוק בשביל הדברים האלו.

בשביל ליצור [sequence](#) יש להשתמש בפקודה הבאה:

```
SQL>CREATE SEQUENCE first_index;↵
```

וניצור בעצם את האינדקס הראשון שלנו עוד לפני שיש לנו בכלל טבלה.

חשוב להבין משהו על sequence ב Firebird: מסד הנתונים עובד עם transaction, ולכן אם יש רצון לשנות בצורה ידנית את הערך, סביר להניח שזה לא יעבוד אם יש יותר מחיבור אחד למסד הנתונים בו זמנית. כמו כן, לא בריא לשנות ידנית את הערך של sequence ומומלץ לעבוד בדרכים המקובלות בלבד.

אחרי שנוצר ה sequence, אפשר ליצור טבלה שתשתמש בו:

```
SQL>CREATE TABLE First_Table(↵
CON>ID INTEGER PRIMARY KEY NOT NULL,↵
CON>FieldA VARCHAR(10)↵
CON>);↵
```

לאחר יצירת הטבלה, יש צורך לחבר בין הטבלה ל sequence. חשוב להבין כי בניגוד להרבה מסדי נתונים, Firebird מבוסס אירועים! מה הכוונה? ניתן לתת הוראות למסד הנתונים מה לעשות לפני הזנת ערכים ומה לעשות אחרי הזנת נתונים (כמובן שזה עובד על עוד סוגי אירועים כמו מחיקה ועדכון). לפני השימוש באירועים, יש צורך לאפס את ה sequence בצורה הבאה:

```
SQL>SET GENERATOR first_index TO 0;↵
```

הקישור בין האינדקס ל sequence מתבצע בצורה הבאה:

```
SQL>SET TERM !! ;↵
SQL>CREATE TRIGGER Next_Index_for_first_index FOR FIRST_TABLE↵
CON>ACTIVE BEFORE INSERT↵
CON>AS BEGIN↵
CON> IF (NEW.ID IS NULL OR NEW.ID = 0) THEN↵
CON>     NEW.ID = GEN_ID(FIRST_INDEX, 1);↵
CON>END!!↵
SQL>SET TERM ; !!↵
```

היות ובברירת מחדל נקודה פסיק מסיימת פקודה, יש צורך להגיד ל Firebird לשנות את זה. הסיבה היא שאם תהיה נקודה פסיק בשורה הנמצאת בתוך ה Trigger שנוצר, אז הפקודה תרוץ לפני הסיום, ותתקבל שגיאה על כך שהשאילתה לא תקינה. לכן, יש צורך להריץ את הפקודה רק כאשר הסיום שנבחרה על ידי איש ה DBA (במקרה הזה 2 סימני קריאה). זו בעצם המשמעות של הפקודה SET TERM.

לאחר השימוש בפקודה SET TERM, הקוד יוצר Trigger. יש לשים לב ש Trigger הוא דבר מאוד גמיש וניתן לעשות איתו הרבה יותר דברים מאשר ממה שמוצג בדוגמה זו. אף שבהתחלה הוא נראה כמוסיף סיבוך, למעשה ניתן בצורה זו גם להחליט מה קורה עם כל שדה אחר שמוזן לנו לפני שהוא בפועל נשמר, משתנה או נמחק. ניתן למשל לעשות CONCAT לכמה פרמטרים, דבר שחוסכת הרבה קוד בשאילתות הרגילות, ואף מקצרת את פעולות ה INSERT הרגילה שתבצע. אם יש צורך ליצור תיעוד של כל שינוי שהתבצע, אז ניתן להכניס את הקוד לתוך Trigger ולא בתוך תוכנה חיצונית, ואז בעצם גם אם יש עוד תכנות שמתחברות למסד הנתונים, הן לא צריכות אף הן לשכפל קוד של שמירת השינויים שנעשו. וכמובן יש עוד שימושים רבים אחרים שניתן לעשות עם Trigger לפי הצרכים השונים של התכנות המשתמשות במסד הנתונים.

אחרי CREATE TRIGGER, נכתב השם של ה Trigger, במקרה הנוכחי השם הוא Next\_Index\_for\_first\_index. לאחר השם יש ציון שיש צורך להכיל את הפעולה על הטבלה First\_Table.

בהמשך יש הגדרה שאומרת מתי ה Trigger יכנס לפעולה, כאשר במקרה הזה הפעולה תתבצע לפני ההזנה (Insert). ניתן לגרום ל Trigger לעבוד על עדכון ומחיקה, ואף לכתוב קוד לכל מצב בתוכו, אך יש לקרוא את המדריכים הרשמיים לשם כך.

לאחר הגדרת הפעולה, יש הגדרה שמציינת שהקוד עבור ה Trigger מתחיל באמצעות הפקודה AS BEGIN.

הקוד בודק עכשיו האם יש ערך כלשהו לשדה ID בטבלה ברשומה הנוכחית, ורק אם אין, יועלה הערך וישמור את הערך החדש ב sequence שהוגדר. יש לשים לב שה sequence ב"כ שומר

את הערך האחרון שהוזן. הפרמטר השני של הפונקציה GEN\_ID אומר בכמה להעלות את הערך של ה-ID. במקרה הזה, נאמר להעלות באחד, אבל חשוב להבין שאפשר אפילו ליצור פעולות מתמטיות שונות בשביל לקבל ID בעל ערך אחר.

לאחר מכן, הסתיים הקוד ליצירת ה-Trigger, ומצב סיום הפקודה חזר למצב הקודם את סיום שורת הפקודה.

על מנת להוסיף ערכים לטבלה, ניצור שאילתת INSERT :

```
SQL>INSERT INTO First_Table (FieldA) VALUES ('Firebird');↵
```

פעולת select פשוטה יכולה להציג את הערכים שהוזנו לטבלה.

בשביל לראות את מבנה הטבלה שנוצרה והקישור שלה אל ה-Trigger ניתן לעבוד עם הפקודה show table :

```
SQL>SHOW TABLE First_Table;↵
```

המבנה לטבלה יוצג בצורה הבאה:

```
ID                INTEGER Not Null
FIELDA            VARCHAR(10) Nullable
CONSTRAINT INTEG_2:
Primary key (ID)

Triggers on Table FIRST_TABLE:
NEXT_INDEX_FOR_FIRST_INDEX, Sequence: 0, Type: BEFORE INSERT, Active
```

## עבודה עם Blob

מהוא Blob ? [האגדה](#) מספרת שבמקור Blob לפני שקיבל את ראשי התיבות של Binary Large Object או Basic Large Object, לא היה פירוש לשם, אלא הוא נלקח מסרט בשם The Blob.

התפקיד של Blob הוא לשמור מידע אשר שום טיפוס נתונים אחר לא יכול לשמור. השמירה נעשית על ידי בתים במקום על ידי סוג תוכן. ולכן בד"כ המידע בתוך Blob אינו מפורש, וב-Firebird למעט טקסט, הוא גם אינו מקובץ. מומלץ לא להשתמש בשדה Blob כדי לשמור מידע בינרי, אבל כאשר יש צורך בכך, Firebird יודע לעבוד עם Blob בצורות רבות.

בניגוד ל-MYSQL, ל-Firebird יש הגדרה אחת של Blob, וההגדרה הזו משתנה בהתאם לסוג התוכן שאנחנו רוצים להכיל על ידי הגדרת סוג תת תוכן. בניגוד למסדי נתונים רבים, Firebird מאפשר למפתחים לבחור את סוג ה-Blob וגם ליצור סוגים חדשים שלו בהתאם לצורך של כל שדה.

ישנם 3 סוגים עיקריים של תת סוג של Blob בדיאלקט 3 של Firebird:

1. מידע בינרי – שומר מידע על בסיס בתים (BLOB - Binary Large Object BLOB)

2. מידע טקסט (CLOB - Character Large Object BLOB SUB\_TYPE TEXT)

3. מידע טקסט מבוסס קידוד שפה (NCLOB - National Character Large Object BLOB (<SUB\_TYPE TEXT CHARACTER SET <national

למרות הרשימה למעלה, נכון לגרסה 2.1.1 של Firebird (הגרסה האחרונה בזמן כתיבת תיעוד זה), ישנם 9 תתי סוגים של Blob, אותם ניתן למצוא מוגדרים בקובץ types.h בקוד המקור של מסד הנתונים:

```
TYPE("BINARY", 0, nam_f_sub_type)
```

```

TYPE("TEXT", 1, nam_f_sub_type)
TYPE("BLR", 2, nam_f_sub_type)
TYPE("ACL", 3, nam_f_sub_type)
TYPE("RANGES", 4, nam_f_sub_type)
TYPE("SUMMARY", 5, nam_f_sub_type)
TYPE("FORMAT", 6, nam_f_sub_type)
TYPE("TRANSACTION_DESCRIPTION", 7, nam_f_sub_type)
TYPE("EXTERNAL_FILE_DESCRIPTION", 8, nam_f_sub_type)
TYPE("DEBUG_INFORMATION", 9, nam_f_sub_type)

```

במידה שנוצר שדה Blob ללא הגדרה של תת תוכן או שיש שימוש בתת מסוג מספר 0, ברירת המחדל תהיה לשמור בתים כמו שהם בתוך השדה. ללא התייחסות לתוכן.

במידה שנוצר שדה Blob עם תת תוכן מספר 1, ההתייחסות לתוכן תהיה של טקסט. זה בד"כ מקביל ל"תזכיר" או תוכן של טקסט. בניגוד ל VARCHAR, אין הגבלה ל 32 קילו בית ולכן כאשר צריך מידע גדול של טקסט, נוח יותר להשתמש בו מאשר שדה של VARCHAR.

במידה שנוצר שדה Blob עם תת תוכן 2, ההתייחסות לתוכן תהיה של מידע SQL שהודר למצב בינרי. ניתן בצורה הזו לשים למשל Stored Procedures ועוד... סוג המידע נקרא BLR או Binary Language Representation.

במידה שנוצר שדה Blob עם תת תוכן 3, ההתייחסות של התוכן תהיה של Access Control List, אשר אינה בשימוש ונשמרת עבור תמיכה אחורה. עד גרסה 4 של Interbase, השימוש ב ACL היה הצורה לבצע אבטחה במסד הנתונים.

במידה שנוצר שדה Blob עם תת תוכן 4, ההתייחסות לתוכן תהיה של רשימות, מערכים (זו לא הדרך לנהל מערכים במסד הנתונים), דפי מידע ועוד.

במידה שנוצר שדה Blob עם תת תוכן 5, ההתייחסות לתוכן תהיה של תוכן מהודר או התייחסות למידע אשר נשמר בצורה הזו בשביל למנוע את השימוש בפעולות JOIN.

במידה שנוצר שדה Blob עם תת תוכן 6, ההתייחסות לתוכן תהיה של מבנה פיזי של טבלה והצורה שהיא פרושה.

במידה שנוצר שדה Blob עם תת תוכן 7, ההתייחסות לתוכן תהיה של ביצוע טרנזאקציות ובעצם מכילה תיאור של מסדי נתונים וההתקשרות בניהם.

במידה שנוצר שדה Blob עם תת תוכן 8, ההתייחסות לתוכן תהיה של איך לשמור מידע על קובץ חיצוני. עד כמה שידוע לי בזמן כתיבת מדריך זה, סוג ה Blob הזה אינו נמצא בשימוש.

במידה שנוצר שדה Blob עם תת תוכן 9, ההתייחסות לתוכן תהיה של מידע לניפוי שגיאות.

כמו שניתן להבין, רוב סוגי ה Blob הם בעצם נבנו לשימוש פנימי של מסד הנתונים, אבל כולם יכולים להשתמש בהם במידה ויש צורך לכך.

בשביל ליצור שדה שהוא מטיפוס Blob יש לכתוב את ההגדרה בצורה הבאה:

```
MyTextBlobField Blob sub_type TEXT NOT NULL
```

ההגדרה תיצור שדה מסוג Blob ותת סוג טקסט, כאשר השדה אינו יכול להיות NULL.

דרך נוספת ליצור תמיכה ב Blob תהיה השימוש במספר הסוג במקום בשם הסוג, כלומר:

```
MyTextBlobField2 Blob sub_type 1 NOT NULL
```

ההגדרה של MyTextBlobField2 זהה לגמרי ל MyTextBlobField, כאשר ההבדל הוא בשימוש במספר הסוג מול השם שלו.

כפי שנאמר מקודם, ניתן ליצור גם תת סוג של Blob בצורה עצמית. בשביל ליצור שדה מוגדר אישית, חובה להשתמש בתת סוג עם מינוס כאשר הטווח הוא עד -32,678. חשוב לדעת שכאשר יוצרים תת סוג עצמאי, חובה לשים את אותו סוג מידע בכל שימוש בשדה הזה, כלומר כל רשומה העובדת עם השדה מותאם אישית, חייבת להיות אותו סוג מידע. במידה ויוזן מידע טקסטואלי פעם ראשונה ופעם שנייה מידע בינרי, מסד הנתונים לא ידע לעבוד עם זה כמו שצריך.

הגדרה של שדה מותאם אישית תראה כך:

```
Bitmap Blob sub_type -1,
PNG Blob sub_type -2,
```

וכך ניתן לשמור מידע של Bitmap ושל PNG עם סוג Blob מותאם אישית (כמובן שלא מומלץ לשמור מידע בינרי בתוך blob).

לא רק שניתן להגדיר תת סוג מותאם אישית, אלא ניתן גם להגיד מה יהיה גודל החוצץ שיקרא את המידע בתוך ה Blob בכל פעם. בברירת מחדל, גודל החוצץ הוא 80 בתים (גודל של מסך טקסט), אבל ניתן להגדיר לשדה גודל אחר. ההגבלה היא עד 32,767 בתים.

הגדרת גודל החוצץ נעשית בצורה הבאה:

```
MyData Blob sub_type TEXT SEGMENT SIZE 1024
```

ההגדרה למעלה מגדירה שגודל החוצץ יקרא כל פעם 1024 בתים של מידע מ MyData.

עוד עובדה שחשוב מאוד לדעת על Blob הוא שאי אפשר להזין בצורה טבעית מידע בינרי למסד הנתונים, אלא צריך כלי חיצוני שיודע לדבר עם מסד הנתונים בשביל לשמור ולקרוא את התוכן של Blob. כלים כאלו הם בד"כ שפות תכנות או כחלק מ User Defined Functions (מוכר גם בשם UDF).

## יצירת כינויים לטיפוסי נתונים

Firebird מאפשר ליצור כינוי לטיפוסי נתונים ובכך לתת שמות בעלי עניין לדרך שנתייחס לטיפוס נתונים. למשל בשביל ליצור טיפוס בוליאני, ניתן לקחת את Smallint כמו שהוא, או ליצור טיפוס בשם Boolean שיקבל 0 או אחד. הכלי שמאפשר לעשות זאת ב Firebird נקרא Domain והוא מסוגל לספק הרבה יותר מאשר רק כינוי עבור הטיפוס שלנו. Domain מסוגל לעזור לוודא שהערך שמתקבל נמצא רק בערכים שיש רצון לקבל אותם (במקרה הבוליאני: 0 או 1). במידה שהערך אינו מתאים להגדרה שהוזנה לו, תתקבל על כך הודעת שגיאה.

במידה ויש צורך ב varchar באורך של 64 תווים לדוגמה (נגיד עבור מספרי טלפון, אז האורך המקסימלי לפי התקן הוא 64 ספרות), אז אפשר ליצור Domain שנקרא לו Phone והוא יהיה varchar באורך 64 תווים. וכך תמיד יהיה אפשר להשתמש ב Phone בשביל לשמור מספרי טלפון.

אבל זה ממש לא כל מה ש Domain מסוגל לעשות. אפשר להגיד לו להזין למשל את שם המשתמש שכרגע דרכו אנחנו מחוברים למסד הנתונים, ניתן להגדיר מימדים למערכים, ניתן להגיד לו שרק ערכים בטווח מסוים יכולים להיכנס (נגיד מחרוזת באורך 3 תווים ומעלה בלבד, או מספר שגודל מ 100 או שמידע לא מכיל תוכן מסוים, או שמידע חייב להיות NULL), אפשר להגדיר לו את הקידוד של הטקסט ועוד אפשרויות רבות אחרות.

איך יוצרים Domain? הגדרה של כינוי פשוט תראה כך:



```
SQL>CREATE DOMAIN Boolean AS SmallInt;↵
```

אבל זה לא מספיק כאשר יוצרים משתנה בוליאני, היות והוא צריך להכיל 0 או 1. לכן את הטיפוס שלנו בצורה בה הוא יוכל להכיל רק 0 או 1:

```
SQL>CREATE DOMAIN Boolean AS SmallInt↵
CON>DEFAULT 0↵
CON>CHECK (value in (0, 1))↵
CON>;↵
```

עכשיו כאשר יהיה שימוש בטיפוס של Boolean, אם יוזן המספר 2, תתקבל שגיאה מהשרת היות והוא יכול לקבל ערכים רק של 0 או 1.

השורה של DEFAULT אומרת מה מחפשים בתור ברירת מחדל, במקרה הזה יש צורך פיזי של ערך ש SmallInt יכול להכיל, ולכן ברירת מחדל היא 0 (כלומר false). לא חובה להשתמש בשורה של ה DEFAULT, אבל היא מומלצת.

חשוב לדעת שבפועל יש טיפוס נתונים בשם Boolean ב Firebird, ויצירה של Domain תשכתב את ההתנהגות של הטיפוס Boolean, וזה עוד דבר ש Domain יודע לעשות ← לשנות התנהגות של טיפוסים משתנים קיימים.

כאשר יש צורך ליצור [Blob](#) שהוא מסוג טקסט עם הקידוד של UTF-8 בכל מצב. אפשר ליצור את ה Domain בצורה הבאה:

```
SQL>CREATE DOMAIN NATIVE_TEXT AS↵
CON> BLOB SUB_TYPE TEXT SEGMENT SIZE 80↵
CON>CHARACTER SET UTF8;↵
```

בהגדרת ה Domain נוצר טיפוס מסוג Blob עם תת סוג של טקסט, וקידוד של UTF-8. בצורה הזו ניתן לכפות בעצם קידוד גם אם הטבלה/מסד נתונים אינו מקודדת בברירת המחדל בתור UTF-8. המידע שישמר בשדה המשתמש ב NATIVE\_TEXT עדיין תישמר כ UTF-8.

אחת התכונות החשובות שיש ל Firebird היא התמיכה באירועים. מה זה אירוע? אירוע הוא סוג של הודעה שהשרת מוציא אל הלקוח כאשר דבר מה התרחש. האירועים האלו לא נפוצים במרבית מנועי מסדי הנתונים בעולם ולכן הרבה מאוד מפתחים שעובדים עם מסדי נתונים לא ייגשו לעבודה עם מסד נתונים בגישה בכלל של שימוש באופציה שכזו. המשתמש יכול ליצור אירועים שונים אודות דברים שונים, כאשר כל מה שמשתנה זה השם של האירוע. שם האירוע מוגבל ל 127 תווים, ובניגוד לשאר השמות ב Firebird יש הבדל בין אותיות גדולות וקטנות (case sensitive).

ניתן ליצור אירועים בשתי צורות:  
1. שימוש ב Triggers:

```
CREATE TRIGGER tr1 FOR employee
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
    POST_EVENT 'new_employee';
END
```

האירוע מתרחש כאשר נוסף עובד חדש לטבלה.

```
CREATE TRIGGER tr2 FOR employee
ACTIVE AFTER UPDATE POSITION 0
AS
BEGIN
    IF (new.PHONE_EXT <> old.PHONE_EXT) THEN
        POST_EVENT 'phone_ext_changed';
    END
```

אירוע אחר מתרחש כאשר מספר הטלפון של העובד השתנה.

2. שימוש ב Stored Procedure

```
CREATE PROCEDURE send_times(event_count integer)
AS
DECLARE VARIABLE nr INTEGER;
BEGIN
    nr = 1;
    WHILE (nr < event_count) DO
        BEGIN
            POST_EVENT 'MY_EVENT';
            nr = nr + 1;
        END
    END
```

יצירת אירוע כאשר הפרוצדורה send\_times מתרחשת, תוך ספירה של כמות הפעמים שהתבקשנו לגרום לאירוע להתרחש.

```
CREATE PROCEDURE send_custom(event_name varchar(127))
AS
BEGIN
    POST_EVENT event_name;
END
```

יצירת Stored Procedure אשר מסוגלת לשלוח כל אירוע המתרחש במערכת.

כמו שהוזכר מקודם, שמות של אירועים מסוגלים להיות עד 127 תווים, ולכן `send_custom` מסוגל להתמודד עם כל שם של אירוע.

הלקוחות שיקבלו את האירועים, הם לקוחות אשר נרשמו מבעוד מועד לקבל את האירוע כאשר הוא מתרחש. הלקוחות יקבלו את האירוע רק לאחר סיום ביצוע ה `commit` על המידע, בגלל ש Firebird מבוסס כאמור על טרנזאקציות.

השימוש באירועים יכול להתבצע על כל סוג פעולה הקיימת במסד הנתונים וגם אירועים אשר לא אמורים להתרחש, יכולים לקבל האזנה אליהם על ידי הלקוחות. הסיבות להשתמש באירועים הן מגוונות. למשל יש "שדון" (`daemon`) אשר צריך לעשות דבר מה כאשר נכנס מידע חדש למערכת. אז במקום לבדוק כל פרק זמן מסוים אם נכנסה רשומה חדשה, אפשר במקום זאת לקבל אירוע ורק כאשר האירוע מתקבל ניתן לגשת למסד הנתונים בשביל לבצע את הפעולה.

יש 2 דברים שחוזרים עם אירוע:

1. שם האירוע
2. כמות הפעמים שהוא התבצע - כלומר אם האירוע עכשיו התרחש 5 פעמים לפני ה `commit`, הכמות שתתקבל תהיה 5. במידה שהאירוע התרחש פעם אחת, תתקבל הספרה אחת לכמות האירועים. הכמות היא כמות של ביצוע עד לפעולת ה `commit` עצמה.

בשביל להאזין לאירוע יש צורך להשתמש ב `Event Manager` אשר עובד ב 2 גישות:

1. סינכרונית
2. א-סינכרונית

אירוע סינכרוני אומר שהאפליקציה תעצור ותחכה לאירוע, וכל עוד האירוע לא מתרחש היא לא תעשה כלום חוץ מלחכות לאירוע. גישה זו טובה כאשר האפליקציה צריכה לבצע פעולות רק על בסיס אירועים וכאשר אין שום אירוע, היא לא אמורה לעשות כלום. למשל, כאשר לקוח החליט לרכוש מוצר, נשלחת הודעה אל חברת כרטיסי האשראי לבדוק האם כרטיס האשראי שלו מסוגל לעמוד במחיר של המוצר, ובכך לאשר או למנוע את רכישת המוצר. עוד אפשרות היא, כאשר משתמש מבקש לקבל מידע בדוא"ל, אפליקציה שצריכה לשלוח את הדוא"ל תחכה לאירוע ותשלח את הדוא"ל לאותו משתמש.

אירוע א-סינכרוני אומר שהאפליקציה לא מחכה לאירוע אלא ממשיכה לעבוד וכאשר מתרחש אירוע אז בזמן כלשהו האירוע מתקבל אל האפליקציה לביצוע. השימוש באירוע בצורה א-סינכרוני טוב כאשר צריך לדעת מתי אירוע מתרחש, אבל צריך להמשיך לבצע עוד פעולות.

## חריגות מותאמות אישית

ב Firebird אפשר ליצור חריגות מותאמות אישית אשר ניתן להרים אותן בתוך Stored Procedure בהתאם לבעיה.

החריגות מקבלות שם ותיאור טקסטואלי אשר יכול להכיל עד 78 תווים. אולם, לא מומלץ להשתמש בחריגות עם טקסט, אלא לתת להם מזהה כלשהו במקום הטקסט אשר יתורגם לשפה הרצויה בצד הלקוח.

מתי יש צורך להשתמש בחריגות? ובכן יש כמה סיבות, אבל אצור תרחיש אחד בשביל ההדגמה: נגיד שיש צורך לקבל הפנייה לרשומה קיימת במסד הנתונים (כדוגמת אינדקס של רשומה), אבל בפועל לא התקבלה הפנייה כזו. אז במקום לקבל שגיאה כללית יותר ממסד הנתונים, אפשר לשלוח חריגה מותאמת אישית בנושא וכך ניתן למפות בעיות טוב יותר בקוד שמטפל בחריגה. עצם השימוש בחריגות (בנוסף לאירועים) הופכת את מסד הנתונים לכלי אשר יכול ליצור API כחלק מהעבודה עם מסד הנתונים במקום להוציא את העבודה החוצה לשפות חיצוניות שצריכות כל אחת בעצמה ליצור קוד שיטפל בזה.

על מנת ליצור חריגה יש לכתוב את הקוד הבא:

```
SQL>CREATE EXCEPTION MyException 'My Exception was raised.';
```

כמו שניתן לראות, מאוד פשוט ליצור חריגות. החריגה שהוגדרה נקראת MyException אשר מכילה טקסט שנכתב עבורה.

בשביל להרים את החריגה עבור השרת, יש לכתוב את הקוד הבא:

```
EXCEPTION MyException;
```

והשרת ירים את החריגה כאשר הוא יגיע לשורה הזו.

ניתן גם להרים חריגות כאשר שגיאות SQL אמיתיות המדווחות על ידי השרת בצורה הבאה:

```
WHEN SQLCODE -902 DO
  BEGIN
    EXCEPTION StatementException;
  END

WHEN EXCEPTION my_error DO
  BEGIN
    EXEPTION my_error 'User defined error raised';
  END

WHEN ANY DO
  BEGIN
    ROLLBACK;
    EXCEPTION;
  END
```

קוד ה SQL למעלה תופס חריגות גם של שרת מסד הנתונים וגם שהוגדרו בעבר ומשתופלות בהתאם להחלטה. ניתן לראות שכאשר קיבלנו הודעת שגיאה 902- מהשרת, אנחנו מעלים חריגה מותאמת אישית לאוויר. כאשר קיבלנו חריגה שהוגדרה על ידי משהו אחר (חריגה שלא קשורה לשרת), יש טיפול גם בה. ואשר כל חריגה הגיעה, דבר ראשון נעשה ROLLBACK ואז

הועלתה החריגה חזרה לאוויר.

עצם היכולת של התפיסה של החריגה ועבודה איתה מספקת אפשרות להתמודד עם בעיות בתוך השאלות, פרוצדורות וtriggers, ובכך להצליח להתמודד עם בעיות ללא קריסה או התנהגות בלתי צפויה.

את הקוד של WHEN אפשר לשים גם תחת פרוצדורות וגם תחת Triggers כך שגם ניתן להתמודד טוב יותר עם בעיות כי אפשר ליצור קוד מותאם טוב יותר עבור כל בעיה שאפשר להיתקל בה.

Firebird תומך בתחביר של View. לפני שאציג כיצד זה עובד, רצוי להסביר מה זה View. זה כלי שלוקח שאילתת select ומפשט אותה לפקודה בודדה. התפקיד שלו זה לספק אפשרות למתן עבודה עם שאילתות מסובכות, לא על ידי כתיבה שונה, אלא על ידי חוסר הצורך בחזרה של הכתיבה. פעולות ה View בעצם גורמות למסד הנתונים לחשוב שמדובר בטבלה חדשה, ולא בכלי ששואב מידע מטבלה.

השימוש העיקרי ב View הוא לשם אבטחה. נגיד ואסור לאפשר למשתמש ספציפי לעשות פעולות select על שדה מסוים, או על טבלה (חשוב לדעת ש Firebird יודע לנעול שדות מסוימים מפני עבודה של משתמשים), אבל אותו משתמש עדיין צריך לקבל מידע מסוים שהוא צריך, אז אפשר לשלוט באיך הוא מקבל אותו, על ידי עבודה עם view. היות ש view מסוגל לעשות כל דבר (למעט פעולת order by) ש select מסוגל (הוא מריץ בתוכו שאילתת select), אפשר גם להגביל את כמות המידע שחוזר לאותו משתמש.

ההבדל בין View לבין Stored Procedure הוא בצורה שהשרת מתעסק איתם. בעוד שעם Stored Procedure הקוד "מהודר" ונשמר בתוך cache כאשר רק הנתונים משתנים, ב View בסה"כ ניגשים לטבלה אחת (או טבלאות אחדות) ומושכים משם מידע. בנוסף, כמו שהזכרתי למעלה, עבור מסד הנתונים, ה View הוא כמו עוד טבלה במערכת, ולכן אין לו שום יתרון על הנתונים שהוא מספק לנו אל מול שאילתת select רגילה על טבלה. בנוסף, View בניגוד ל Stored Procedure מאפשר לנו לבצע רק פעולות select ולא פעולות אחרות.

ב view אין אפשרות לבצע אופטימיזציה של השאילתות, ובנוסף אופן השימוש בו גורם לכך שהשימוש ב view יתבצע בצורה אטית יותר מאשר שימוש ב select או stored procedure רגילים, ולכן צריך לקחת בחשבון פרמטרים רבים כאשר צריך לבחור אם להשתמש ב view או לא.

ניתן להשתמש ב 2 צורות עם View, הראשונה היא מאוד פשוטה:

```
CREATE VIEW name AS
SELECT User, Email FROM Users WHERE Permissions >= 700
;
```

השאילתה תחזיר את כל השמות והדוא"ל של המשתמשים עם הרשאות של 700 ומעלה. חשוב לדעת שהנקודה פסיק אומרת לסיים את ה view ולא את פעולת ה select. ניתן גם להוסיף את השורה:

```
WITH CHECK OPTION
```

כלומר:

```
CREATE VIEW name AS
SELECT User, Email FROM Users WHERE Permissions >= 700
WITH CHECK OPTION;
```

השורה שהוספנו אומרת לשרת לנעול את הטבלה/טבלאות לשינויים, תוספות ומחיקות כאשר אנחנו קוראים ל View.

אפשרות נוספות להגדרת View היא על ידי הוספת פרמטרים שאיתם נרצה להשתמש:

```
CREATE VIEW name2 (User, Email) AS
SELECT User, Email FROM Users WHERE Permissions >= 700
WITH CHECK OPTION
```

התחביר של ה View שלנו זהים לגמרי בהתנהגויות שלהן, רק בדוגמה השנייה מוגדר בצורה

מוגדרת ל view מה הן השדות שיכולות להיות בשימוש. במידה שנשתמש ב view השני, יש חשיבות רבה מאוד לשם השדות. אי אפר לחזור על 2 שמות ברשימת השדות.

כדי להשתמש ב view השימוש מתבצע בפעולת select רגילה:

```
select user, email from name;
```

על המידע שחוזר, אפשר כמובן ולעשות עוד פעולות בתוך ה select שמריץ את ה view.

## הרשאות משתמשים

במידה ונוצרו משתמשים שונים עם [gsec](#) אפשר להחיל עליהם הרשאות. ההרשאות עצמן מוגדרות לפי מסד הנתונים הספציפי בו יש עבודה ולא לפי מסד הנתונים של המשתמשים, ולכן אפשר ליצור משתמש אחד עם סוגי הרשאות שונים בהתאם למסד הנתונים אליו הוא מחובר. בצורה רגילה אפשר ליצור הרשאות בצורה הבאה<sup>1</sup>:

```
SQL>GRANT select ON first_table TO username WITH GRANT OPTION;↵
```

במידה שיש הרבה משתמשים שונים, להחיל כל הרשאה כזו לאותם משתמשים לא תהיה דבר של מה בכך, ולכן המציאו את roles. התפקיד של roles הוא ליצור קבוצות של הרשאות, ועליהם אנחנו מכילים את ההרשאות עצמן.

```
SQL>CREATE ROLE my_role;↵
```

זו כל ההגדרה של role. ועכשיו יהיה אפשר לשים הרשאה על my\_role במקום על משתמש ספציפי:

```
SQL>GRANT select ON first_table TO my_role WITH GRANT OPTION;↵
```

אחרי שישנם קבוצות עם הרשאות, ניתן להכיל את הקבוצות על המשתמשים בצורה הבאה:

```
SQL>GRANT my_role TO username;↵
```

ועכשיו המשתמש שייך ל role שהגדר.

---

1 לא אסביר כאן את התחביר של [GRANT](#) ו [REVOKE](#). במידה ואין לכם הכירות איתה מומלץ לקרוא את התיעוד בנושא.



## שמירת תוכן של הפלט

כאשר משתמשים ב `isql`, ניתן להגדיר ל `isql` לשים את התוצאות של `select` (או כל פעולה אחרת) בקובץ טקסט במקום על המסך, ובכך יהיה לנו קל יותר לעקוב אחרי פעולות שנעשו והתוצאות שלהן.

על מנת להגדיר ל `isql` להכניס לקובץ את הפלט, יש להשתמש בפקודה `out` (אשר שייכת ל `isql` ולא לשפת `sql`):

```
SQL>OUT /tmp/output.txt;↵
```

עכשיו כאשר הורצה שאילתה או כל פעולה אחרת, התוכן יכנס אל הקובץ `output.txt` אשר נמצא בספריית `tmp`.

אפשר גם להגיד לפלט איך להפריד בין שדות:

```
SQL>select application, ', ', package, ', ', vendor, ';' from installer;↵
```

התוצאה תהיה:

```
application      package          vendor
=====
Firebird ,      Firebird Super , Firebird Foundation ;
```

בשביל להחזיר את הפלט חזרה למסך, יש להשתמש בפקודה `OUT` בשנית:

```
SQL>OUT;↵
```

במדריך זה הצגתי חלק מאוד קטן מהיכולות והתכונות של Firebird אשר לדעתי יעזרו לכם להתחיל ולעבוד עם מסד הנתונים.

המדריך אינו מגיע בתור תחליף עבור המדריכים הרשמיים של מסד הנתונים, אלא בתור כלי התייחסות בשפה העברית, אשר יעזור לאחר קריאתו להבין טוב יותר את המדריכים הרשמיים.

אני מקווה שהמדריך עזר לעורר בכם עניין במסד הנתונים, מספיק בשביל להתחיל לחקור אותו ולגלות עוד מסד נתונים קוד פתוח אשר יכול לתרום הרבה מאוד לניהול מידע שלכם.

ברצוני להודות לכל האנשים שעזרו לי עם מדריך זה:

- אור
- אסף ספיר
- יחזקאל ברנט
- מאיר קריחלי
- שלומי פיש

וכל מי שסייע לי ולא הוזכר בשמו/כינויו