# Software Documentation

## DB-Structure

סכימת הDB מחזיקה שני סוגי חומר:

1. הData עצמו שנטען מYAGO ועליו נשאלות השאלות.
2. טבלאות Meta ששומרות נתונים על משתמשים ומשחקים.

טבלאות Meta

ישנן שתי טבלאות Meta – User ו-Score. User היא טבלה המתארת שחקן, היא מכילה את פרטי החיבור שלו (id, name, password) ומידע סיכומי עליו (לדוגמא מספר משחקים ששיחק). הטבלה Score מתארת תוצאת משחק.

טבלאות Data

בחלק זה של הסכימה ישנן שתי סוגי טבלאות:

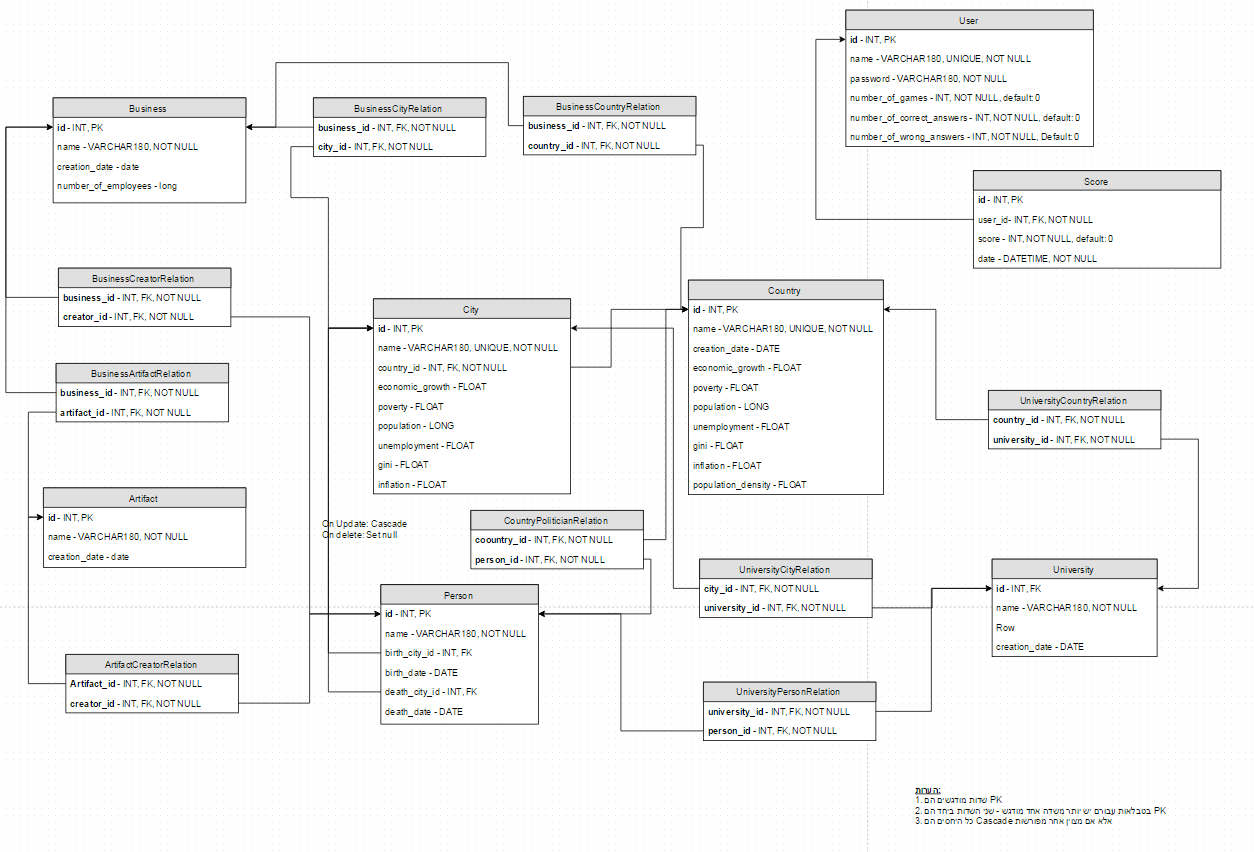
1. טבלאות המתארות entity.

כל entity הוא שורה בטבלה הרלוונטית (לדוגמא הentity Israel הוא שורה בטבלה Country). לentity יש id רץ, שם, פרטים נוספים עליו (לדוגמא, עבור עסק מתי הוא נוצר), וFK לאובייקטים אחרים בסכימה.

רצינו שבאמצעות FK נוכל להגיע מכל entity לentity מסוג country שכן השאלות נשאלות על ארצות.

1. טבלאות המתארות קשר Many-To-Many בין שני entities. טבלאות אלו נגמרות במילה Relation.

בעמוד הבא ניתן למצוא תיאור סכמתי מלא.



## מקורות מידע מYAGO

* yagoTransitiveTypes – מקובץ זה אנחנו מפרסרים את הentities. אנחנו מחפשים entities שיורשים/ממשים typeים מסוימים (לדוגמא על מנת למצוא ארצות אנחנו מחפשים entities שהtype שלהם הוא <wikicat\_Countries>).

הסיבה שאנחנו משתמשים בקובץ הטרנזיטיבי ולא בקובץ yagoTypes הוא בגלל שאנחנו מעוניינים בכל עץ הירושה. לדוגמא עבור city אנחנו נרצה גם entities שממשות את הערים, וגם ערים שממשות את ערי נמל (ערי נמל יורשות מערים).

* yagoLabels – מקובץ זה אנחנו מפרסרים את השם של הentity לאחר שמצאנו אותה מהקובץ הקודם.
* yagoDateFacts – פרסור עובדות תאריכים בנוגע לentities שמצאנו.
* yagoFacts – פרסור עובדות.
* yagoLiteralsFacts – פרסור עובדות.

## Code Structure

### פרסור מידע

פרסור המידע מתבצע ע"י האובייקט DataCollector. האובייקט מחזיק מספר attributes שכל אחד מהם הוא MAP בין string המתאר את הentity (לדוגמא <Israel>) לאובייקט המכיל את כל המידע על הentity.

בתחילה אנחנו עוברים על הקובץ yagoTransitiveTypes ואנחנו מחפשים entities שהtype שלהם הוא מחלקה שהגדרנו מראש. המחלקות הללו הן אלו שנרצה לשאול עליהם שאלות, לדוגמא מדינות, ערים, פוליטיקאים וכדומה.

עבור כל entity שאנחנו מוצאים אנחנו יוצרים אובייקט מהסוג המתאים (לדוגמא עבור הentity <Israel> ניצור אובייקט Country). האובייקט מכיל data members שיחזיקו את כל המידע שאנחנו רוצים לדעת על אותו אובייקט. הdata members הם public כי כל איסוף המידע מתבצע באמצעות reflaction.

לאחר שאספנו את כל הentities אנחנו מפרסרים את השם שלהם מyagoLabels ועובדות עליהם מקבצי העובדות. עבור עובדות המקשרות בין שני entities אנחנו גורמים לentity אחד להצביע על האחר (מהסיבה הזאת הdata collector מחזיק MAP).

לאחר שאספנו את כל המידע על כל הentities, אנחנו מבצעים תהליך של post process. התהליך נועד למחוק entities שלא נרצה להעלות לDB, לדוגמא כאלה שאין עליהם מידע מעניין או כאלה שלא ניתן לקשר לCountry. הpost process חייב להתבצע בסדר מוגדר מראש, כיוון שראשית אנחנו מוחקים entitiy שהם מדינות, ולאחר מכן אנחנו מוחקים את כל מי שאין לו מדינה או שמקושר למדינה הזאת וכן הלאה.

### העלאת מידע לDB

ישנן שתי מחלקות רלוונטיות:

1. DBConnection – זוהי המחלקה שיודעת "לדבר בשפת הDB", וזוהי מטרתה היחידה. היא זאת שעושה בפועל את העבודה, מכירה את הטבלאות בסכימה, מפעילה את השאילתות על הdb וכדומה.

במידה והתעוררה שגיאה המתודות זורקות DBException.

1. DAO – זוהי מחלקה שיודעת לקבל פקודות מהלוגיקה ולהפעיל בצורה נכונה את הDBConnection. זוהי השכבה שיוצרת אבסטרקציה בין לוגיקה המערכת לDBConnection. לדוגמא, כאשר נרצה לשלוף מידע על מדינה, ראשית נרצה לבדוק שהמדינה הזאת קיימת ורק לאחר מכן נרצה לשלוף ממנה את המידע, הdao יודע להשתמש במתודות הנכונות מהdbConnection על מנת לבצע את העבודה הנ"ל.

דוגמא נוספת – המחלקה יודעת להתמודד עם מקרים בהם נרצה לשלוף 4 אובייקטים בעלי מאפיין מסוים, אבל לא קיימים בDB 4 אובייקטים המקיימים את התכונה.

הDAO זורק 3 סוגי שגיאות –

* + - DAOException – אם קרתה שגיאה מסוג DBException (לדוגמא החיבור התנתק).
    - DataNotFoundException – אם ביקשנו Data שלא נמצא בDB, לדוגמא שדה שהערך שלו null.

### לוגיקת המשחק

ישנן ארבע מחלקות רלוונטיות:

1. Question – מחלקה זו מייצגת שאלה, מחלקה פשוטה שיודעת להחזיק את השאלה וארבע התשובות שניתנו לה, כאשר מבקשים את התשובות האפשריות היא מחזירה אותן ברשימה מעורבבת. בנוסף לכל אובייקש שאלה שנוצר היא מספקת פונקציה שבודקת האם תשובה נכונה. מחלקה זו אינה עובדת מול שכבת הDAO אלא רק מחזיקה שאלה בזיכרון, בהמשך נראה את מחלקת QuestionFactory שמייצרת את השאלות.
2. User – מחלקה זו מייצגת משתמש, מספקת את היכולת להתחבר עם משתמש, להגדיר למשתמש מדינה מועדפת שהוא ישאל במשחק רק שאלות שקשורות למדינה. ניתן לקבל את התוצאה הנוכחית של המשתמש, ניתן לשמור תוצאה זאת בסטטיסטיקת המשתמש בעזרת גישה לDAO. במהלך ריצת התוכנית בנוסף לשמירת התוצאה בזיכרון אנחנו מעדכנים סטטיסטיקה של המשתמש גם בDB (כמות תשובות נכונות ושגויות, מספר משחקים). למחלקה קיימת גם מתודה סטטית שמאפשרת רישום משתמש חדש.
3. מחלקת QuestionFactory – מחלקה זו היא המחלקה היחידה שמכירה את השאלות שאנחנו שואלים במשחק, עבור כל שאלה היא מחזיקה מתודה שיודעת לייצר את השאלה עצמה. כל מתודה כזאת מבקשת משכבת הDAO את הנתונים שהיא צריכה, המתודות הללו מתחלקות לשתי קבוצות שאלות שקשורות למדינה ספציפית וכאלה שלא. כאשר קוראים ליצור שאלה ניתן להעביר מדינה מועדפת או אם לא מעוניינים null, המפעל מגריל מספר מתאים לפי האם קיימת העדפה או לא ומעביר לאחת המתודות לפי ההגרלה. במידה ומשתמש לא בחר מדינה המפעל מגריל מדינה כדי לאפשר לשאול שאלה גם מקבוצת השאלות הראשונה. תהליך יצור שאלה כולל בדרך כלל שלושה שלבים, קבלת התשובה הנכונה, יצירת שלוש אפשרויות שגויות והחזרת אובייקט חדש מסוג question המכיל את המידע.
4. מחלקת gameRunnner – מחלקה זו היא הפנים של שכבת הcore (business logic), שכבת הUI תפנה אך ורק מחלקה זו. מחלקה זו מספקת את כלל השירותים הנדרשים לכל שכבת UI שתרצה לייצר את המשחק. מחלקה זו מספקת את היכולת לרשום משתמש, למשתמש להתחבר למשחק, להתחיל לשחק, לצבור ניקוד, לקבל שאלה חדשה, לקבל את 10 התוצאות הטובות ביותר, לקבל את כל המדינות עבור בחירת מדינה מועדפת. השכבה תומכת בשימוש של משתמש אחד או שניים (לפי הגדרה שאנו בחרנו). השכבה תבעבע מעלה שגיאות רק במידה ולא ניתן להמשיך לשחק משמע או שיש תקלה מול הDB או שנגמר המשחק וכדי להמשיך לשחק צריך להתחל אותו. במהלך משחק האובייקט תמיד יחזיק שאלה אצלו שניתן לבדוק אם ענית נכון או לא וכך להמשיך לשאלה הבאה.