FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z KIV/DB2

Tradiční piškvorky

Zdeněk Častorál
A19N0026P
zcastora@students.zcu.cz

12. 5. 2020

Obsah

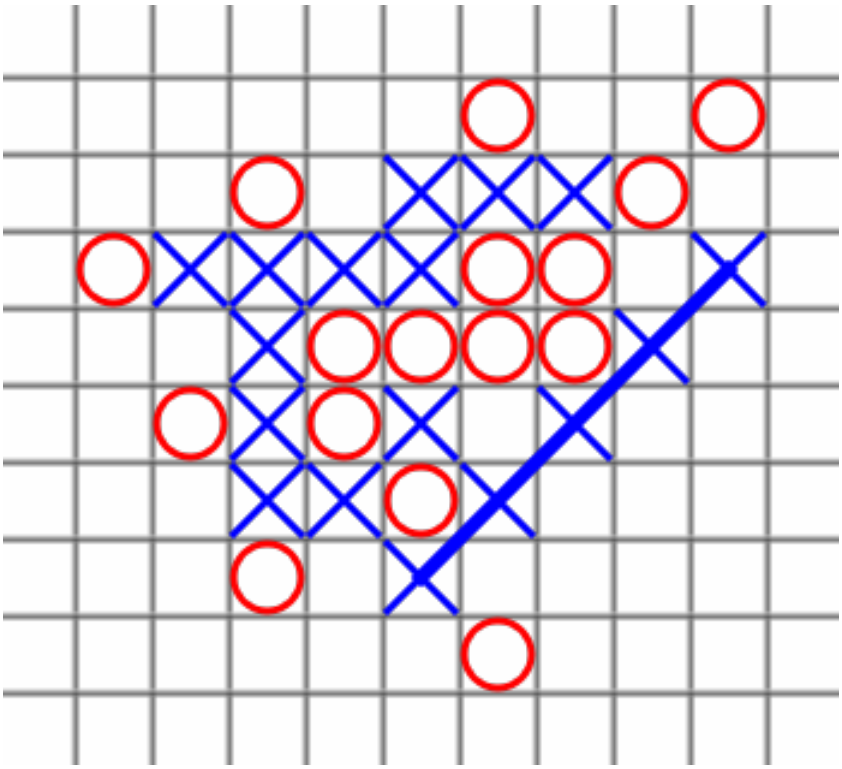
1 Zadání

Cílem této práce je navrhnout a vytvořit relační databázi pro hraní známé strategické deskové hry *Piškvorky*. Ze hry bude řešena pouze databázová vrstva aplikace, proto se snažte co nejvíce programových rutin uložit do databáze a také zajistěte jejich automatické spouštění při nastalé události.

Tradiční piškvorky jsou hra pro dva hráče, která se hraje na čtverečkovaném papíře. Ve hře se hráči střídají po jednom tahu, ve kterém každý hráč umístí na volné místo na papíře svoji značku. Obvykle hráči používají symboly kolečko (O) a křížek (X). Vítězí ten hráč, kterému se podaří sestavit nepřerušovanou řadu alespoň pěti svých značek v libovolném směru (vodorovně, svisle, uhlopříčně). Pokud není možné umístit novou značku, hra končí remízou.

Hra probíhá na papíře, který bude mít definovanou velikost. Nejmenší rozměr papíru, na kterém bude možné hrát, může mít rozměry 5x5 čtverečků, největší rozměr papíru nesmí překročit velikost 20x20 čtverečků. Také bude možné definovat, na kolik značek se bude hrát. Velikost vítězné řady by měla být z intervalu 5 až 15. Je třeba si dát pozor na skutečnost, aby se vítězná řada vešla na definovaný papír.

Také se bude měřit herní čas. Hra začíná umístěním první značky začínajícího hráče. V ten samý okamžik se začíná měřit herní čas druhého hráče a to až do doby, kdy umístí svoji značku. Pak se začíná měřit čas začínajícího hráče. Jakmile začínající hráč umístí svoji druhou značku, pozastaví se měření jeho herního času a opětovně se spustí měření času druhého hráče. A tak pořád dokola, až hra skončí výhrou jednoho z hráčů nebo remízou.



Obrázek 1.1: Piškvorky, Zdroj: `https://cs.wikipedia.org/wiki/Piškvorky`

V relační databázi budou evidována data v těchto tabulkách:

- `OMEZENI` – Tato tabulka slouží jako parametry programu. Obsahuje informace, jak veliký či malý může být čtverečkovaný papír, na kterém se bude hrát a také jak dlouhá či krátká může být minimálně řada symbolů vítězného hráče.
- `STAV` – Číselník ukazující, v jakém stavu se může nacházet hra. Stavy mohou být tyto: rozehraná, vítězství začínajícího hráče, prohra začínajícího hráče nebo remíza.
- `HRAC` – Každý hráč, který bude chtít hrát, musí být zaregistrován. Výhodou bude možnost sledovat jeho statistiky hraní, tj. počet vítězství, proher či remíz, a zda začínal nebo hrál jako druhý.
- `HRA` – Každá hra se hraje na novém čistém papíře o dané velikosti na požadovaný počet vítězných symbolů. Hru hrají dva různí hráči, kde jeden z nich umisťuje kolečka, ten druhý křížky a jeden z nich celou hru začíná. Tabulka bude také obsahovat, v jaké stavu je hra a také herní časy obou hráčů.
- `TAH` – Umístění své značky hráčem, který je na řadě v rozehrané hře. Ke každému tahu se bude automaticky ukládat časová značka, která dočasně nebo trvale zastavuje měření času právě hrajícímu hráči.

Z uložených dat v databázi vytvořte databázové pohledy, které nabídnou tato data:

- `PAPIR` – Zobrazení čtverečkovaného papíru obsahující všechny dosud provedené tahy právě probíhající hry. Každý řádek papíru bude zobrazen voláním funkce `RADEK_PAPIRU`.
- `VYHRY_ZACINAJICI` – Hry, ve kterých zvítězil začínající hráč. Obsahuje parametry hry (rozměry papíru, požadovaná velikost vítězné řady), jména hráčů, kdo začínal (a zvítězil), kdo používal jaké značky, jak dlouho celá hra trvala v sekundách, kolik bylo zahráno tahů.
- `PROHRY_ZACINAJICI` – Hry, ve kterých prohrál začínající hráč. Obsahuje parametry hry (rozměry papíru, požadovaná velikost vítězné řady), jména hráčů, kdo začínal (a prohrál), kdo používal jaké značky, jak dlouho celá hra trvala v sekundách, kolik bylo zahráno tahů.
- `REMIZY` – Hry, které dospěly do remízy. Obsahuje parametry hry (rozměry papíru, požadovaná velikost vítězné řady), jména hráčů, kdo začínal, kdo používal jaké značky, jak dlouho celá hra trvala v sekundách.

V databázových pohledech `VYHRY_ZACINAJICI`, `PROHRY_ZACINAJICI` a `REMIZY` získejte dobu hraní hry jako součet herních dob obou hráčů.

V databázi budou uloženy a používány tyto funkce (s parametry):

- `SPATNY_PARAMETR` – Podle návratové hodnoty funkce poznáme, že je:
  - 0 - vše v pořádku.
  - 1 - příliš malý počet řádků na papíru (menší než 5).
  - 2 - příliš velký počet řádků na papíru (větší než 20).

  - 3 - příliš malý počet sloupců na papíru (menší než 5).
  - 4 - příliš velký počet sloupců na papíru (větší než 20).
  - 5 - příliš malý počet znaků ve vítězné řadě (menší než 5).
  - 6 - příliš velký počet znaků ve vítězné řadě (větší než 15).
  - 7 - vítězná řada delší, než šířka papíru.
  - 8 - vítězná řada delší, než výška papíru.
- `RADEK_PAPIRU` – Vrátí řetězec, který odpovídá konkrétnímu řádku papíru dané hry. Pro výpis zvolte v řetězci tyto symboly:
  - `X` - značka křížek jednoho hráče dané hry.
  - `O` - značka kolečko druhého hráče hry.
  - `mezera` - symbol označující volné políčko na papíře.
- `HERNI_CAS` – Vrátí číslo určující, kolik sekund hrál daný hráč danou hru, tj. sečte rozdíly časových značek, kdy hrál daný hráč a kdy hrál před ním druhý hráč.
- `REMIZA` – Vrátí hodnotu `TRUE`, pokud daná hra dospěla do remízového stavu, tj. není možné udělat další tah. Jinak vrací hodnotu `FALSE`.
- `VYHRA` – Vrátí hodnotu `TRUE`, pokud právě hrající hráč v dané hře vyhrál, tj. svým posledním tahem docílil požadované minimální délky vítězné řady svých značek. Jinak vrací hodnotu `FALSE`.

V databázi budou uloženy a jako těla triggerů používány tyto procedury (s parametry):

- `ZABRAN_HRE` – Zabrání vytvoření nové hry, pokud je nastaven špatně libovolný parametr hry.
- `ZABRAN_TAHU` – Zabrání tahu, který není možno udělat.
- `KONEC_HRY` – Spočítání herních časů hráčů právě dokončené hry.
- `STATISTIKY` – Aktualizace statistických údajů hráčů, kteří dohráli danou hru.

O automatické činnosti v databázi se postarají triggery o:

- hlídání parametrů nové hry (volání funkce `SPATNY_PARAMETR`),
- hlídání hráčů, aby se ve hře pravidelně střídali po jednom tahu,
- hlídání hráče, aby nepokládal svoji značku na již obsazené místo,
- hlídání hráče, aby nemohl realizovat tah ve hře, ve které nehraje,
- hlídání hráče, aby nemohl realizovat tah ve hře, která již skončila,
- hlídání a aktualizace stavu hry, tj. zda nedospěla do remízy nebo vítězství jednoho z hráčů (volání funkcí `REMIZA` a `VYHRA`),
- spočítání herní doby hráčů, kteří dohráli aktuální hru (volání funkce `HERNI_CAS`),
- aktualizace statistických údajů hráčů, kteří dohráli aktuální hru.

Konfigurace, spuštění a průběh hry

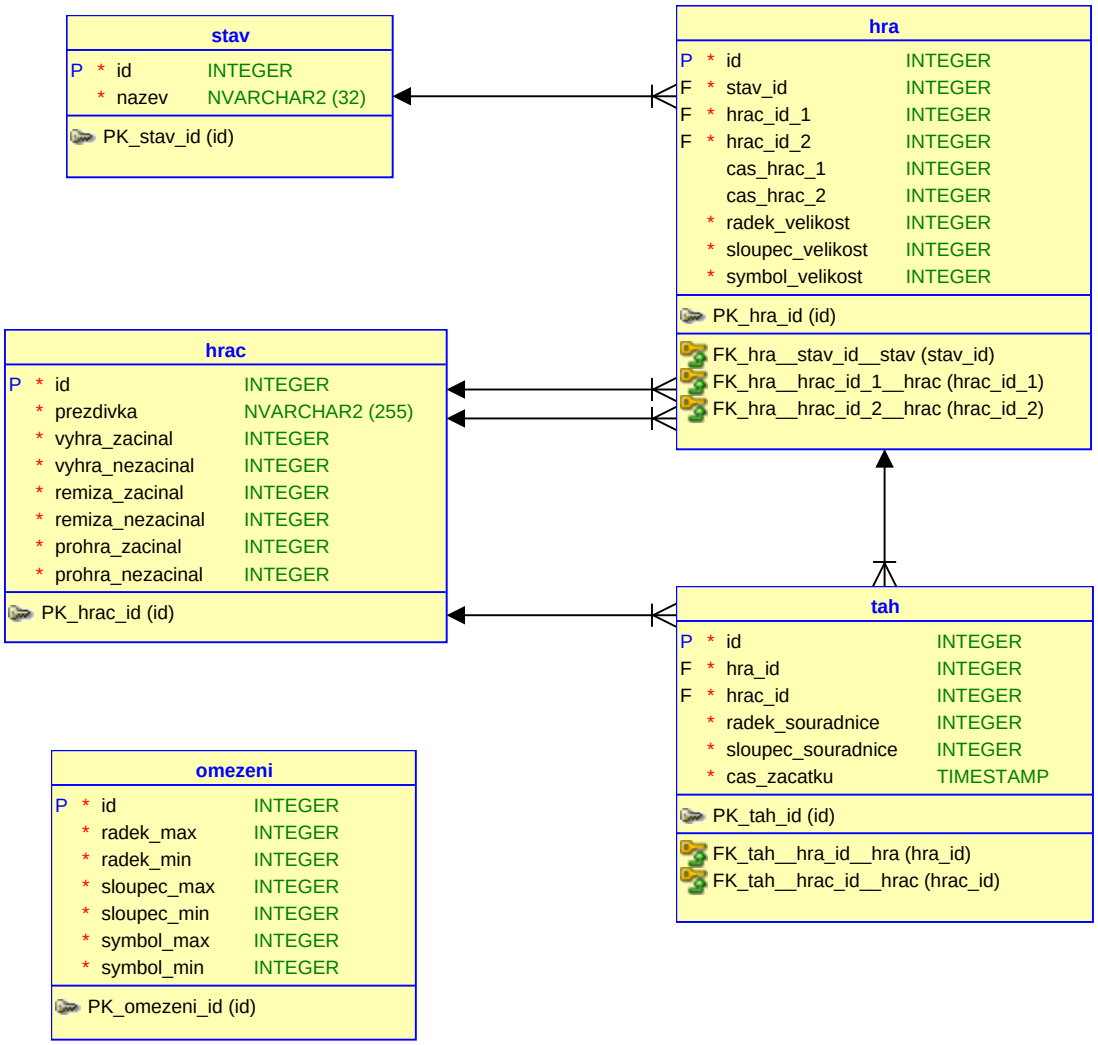
1. Jednorázová konfigurace databáze:
   (a) Naplnění tabulky `OMEZENI` hraničními hodnotami parametrů nové hry.
   (b) Naplnění tabulky `STAV` daty odpovídající různým stavům hry.
2. Registrace hráčů je provedena vkládáním nových záznamů do tabulky `HRAC`.
3. Hra je zahájena úspěšným vložením nového záznamu do tabulky `HRA`. Neúspěšné vložení znamená, že některé parametry hry jsou mimo hraniční hodnoty definované v tabulce `OMEZENI`.
4. Že hra běží, ověříme zobrazením papíru, tj. vypsáním řádků prostřednictvím databázového pohledu `PAPIR` pro aktuálně rozehranou hru.
5. Každý tah ve hře je proveden vložením nového záznamu do tabulky `TAH`. Pokud ke vložení nedošlo, byla aktivována některá z výše uvedených kontrol.
6. Po úspěšném tahu je vhodné si vždy zobrazit aktuální situaci na papíře prostřednictvím databázového pohledu `PAPIR`.
7. Pokud hra neskončila, pokračuj bodem 5, jinak bodem 8 (výhra) nebo bodem 9 (remíza).
8. Hra skončila vítězstvím jednoho z hráčů. To je vhodné ověřit voláním pohledu `VYHRY_ZACINAJICI` nebo `PROHRY_ZACINAJICI` a také se podívat na aktualizované statistiky hrajících hráčů v tabulce `HRAC`. Jdi na bod 10.
9. Hra skončila remízou. To je vhodné ověřit voláním pohledu `REMIZY` a také se podívat na aktualizované statistiky hrajících hráčů v tabulce `HRAC`.
10. Novou hru zahájíme bodem 3, s případnou registrací nového hráče bodem 2.

2 Datový model

Datový model obsahuje pět tabulek. Jedná se o tabulky:

- OMEZENI – tato tabulka slouží jako parametry programu,
- STAV – číselník, který ukazuje, v jakém stavu se může hra nacházet,
- HRAC – tato tabulka obsahuje registrované hráče a jejich statistiky,
- HRA – tato tabulka obsahuje jednotlivé hry (včetně aktuálního stavu hry a herních časů obou hráčů),
- TAH – tabulka, která obsahuje jednotlivé tahy umístěné hráčem, který je na řadě v rozehrané hře.

Detailní specifikace tabulek je uvedena v kapitole 1. Datový model včetně atributů je uveden na obrázku 2.1.



Obrázek 2.1: Datový model

3 Implementace

Implementace byla provedena prostřednictvím jazyka PL/SQL v prostředí databáze Oracle verze 19c.

V této kapitole je popsána implementace jednotlivých databázových objektů, jejichž detailní specifikace a analýza je uvedena v kapitole 1.

3.1 Tabulky

Tabulky byly implementovány v souladu s datovým modelem popsaným v kapitole 2. V kapitole 3.1.1 je podrobněji popsána implementace tabulky TAH.

Skript pro vytvoření všech tabulek je uložen v souboru:

`src\01_create_scripts\01_create_tables.sql`

3.1.1 TAH

Tabulka TAH obsahuje jednotlivé tahy umístěné hráčem, který je na řadě v rozehrané hře. Atribut hrac_id specifikuje hráče, který umístil daný tah ve hře, která je specifikovaná atributem hra_id. Atribut cas_zacatku ukládá časovou značku, která reprezentuje začátek měření herního času oponenta. V definici tabulky je časová značka generována automaticky při vložení nového záznamu prostřednictvím klíčových slov DEFAULT CURRENT_TIMESTAMP.

Implementace tabulky TAH včetně přidání primárních a cizích klíčů zobrazuje ukázka kódu 3.1.

Ukázka kódu 3.1: Vytvoření tabulky TAH

```
1  CREATE TABLE tah (
2      id                    INTEGER NOT NULL,
3      hra_id                INTEGER NOT NULL,
4      hrac_id               INTEGER NOT NULL,
5      radek_souradnice      INTEGER NOT NULL,
6      sloupec_souradnice    INTEGER NOT NULL,
7      cas_zacatku           TIMESTAMP DEFAULT CURRENT_TIMESTAMP
           NOT NULL
8  );
9
10 ALTER TABLE tah ADD CONSTRAINT pk_tah_id PRIMARY KEY ( id );
11
12 ALTER TABLE tah
13     ADD CONSTRAINT fk_tah__hra_id__hra FOREIGN KEY ( hra_id )
14         REFERENCES hra ( id );
15
16 ALTER TABLE tah
17     ADD CONSTRAINT fk_tah__hrac_id__hrac FOREIGN KEY ( hrac_id
           )
18         REFERENCES hrac ( id );
```

3.2 Funkce

V databázi byly implementovány všechny požadované funkce uvedené v kapitole 1. Dále byla implementována pomocná funkce `VYKRESLI_PAPIR`, která slouží k vykreslení všech řádků papíru dané hry a je nutná k implementaci pohledu `PAPIR` (podrobněji viz kapitola 3.4.1). Pro shrnutí, databáze obsahuje funkce:

- `RADEK_PAPIRU` – vrací řetězec, který odpovídá konkrétnímu řádku papíru dané hry,
- `VYKRESLI_PAPIR` – vrací řetězec, který odpovídá všem řádkům papíru dané hry (podrobněji v kapitole 3.2.1),
- `VYHRA` – vrací hodnotu `TRUE`, pokud právě hrající hráč v dané hře vyhrál, jinak vrací `FALSE`,
- `REMIZA` – vrací hodnotu `TRUE`, pokud daná hra dospěla do remízového stavu, jinak vrací `FALSE`,
- `HERNI_CAS` – vrací číslo, které určuje, kolik sekund hrál daný hráč danou hru,
- `SPATNY_PARAMETR` – vrací číslo podle konkrétní chyby v nastavení hry.

Skript pro vytvoření všech funkcí je uložen v souboru:

`src\01_create_scripts\02_create_functions.sql`

3.2.1 VYKRESLI_PAPIR

Jak již bylo zmíněno výše, funkce `VYKRESLI_PAPIR` vrací řetězec, který odpovídá všem řádkům papíru dané hry. Tato funkce je volána v pohledu `PAPIR` (viz kapitola 3.4.1).

Tato funkce pro zadané `id` hry volá ve smyčce funkci `RADEK_PAPIRU` pro každý řádek dané hry. Zřetězení všech výstupních řetězců funkce `RADEK_PAPIRU` je vráceno jako výsledný řetězec funkce `VYKRESLI_PAPIR`. Implementace této funkce je zobrazena v ukázce kódu 3.2.

Ukázka kódu 3.2: Implementace funkce `VYKRESLI_PAPIR`

```
1  CREATE OR REPLACE FUNCTION vykresli_papir ( hra_id IN INTEGER )
       RETURN NVARCHAR2 AS
2  hra_radek hra%ROWTYPE;
3  papir_vypis NVARCHAR2(2100) := '';
4
5  BEGIN
6      SELECT * INTO hra_radek
7      FROM hra
8      WHERE hra.id = hra_id
9          AND rownum = 1;
10
11     FOR aktualni_radek IN 1..hra_radek.radek_velikost LOOP
12         papir_vypis := papir_vypis|| radek_papiru ( hra_radek.
               id, aktualni_radek )|| chr(10);
13     END LOOP;
14
15     RETURN papir_vypis;
16 END;
17 /
```

3.3 Procedury

V databázi byly implementovány všechny požadované procedury uvedené v kapitole 1. Navíc byla implementována procedura `AKTUALIZUJ_STAV_HRY`, která aktualizuje stav dokončené hry, a pomocné procedury `REGISTRUJ_HRACE`, `ZAHAJ_HRU`, `PROVED_TAH`. Databáze tedy obsahuje procedury:

- `KONEC_HRY` – procedura spočítá herní časy hráčů právě dokončené hry (podrobněji v kapitole 3.3.1),
- `AKTUALIZUJ_STAV_HRY` – tato procedura aktualizuje stav dokončené hry (podrobněji v kapitole 3.3.2),
- `STATISTIKY` – aktualizuje statistické údaje hráčů, kteří dohráli danou hru,
- `ZABRAN_HRE` – zabrání vytvoření nové hry, pokud je nastaven špatně některý z parametrů hry,
- `ZABRAN_TAHU` – zabrání tahu, který není možné udělat,
- `REGISTRUJ_HRACE` – pomocná procedura pro registraci hráče (podrobněji v kapitole 3.3.3),
- `ZAHAJ_HRU` – pomocná procedura pro zahájení nové hry,
- `PROVED_TAH` – pomocná procedura pro realizaci tahu.

Skript pro vytvoření všech procedur je uložen v souboru:

`src\01_create_scripts\03_create_procedures.sql`

3.3.1 KONEC_HRY

Procedura `KONEC_HRY` je volána při ukončení hry (tzn. výhře jednoho z hráčů, či remíze) z triggeru `VYHODNOT_STAV_HRY` (viz kapitola 3.5.1). Tato procedura spočítá herní časy zúčastněných hráčů prostřednictvím funkce `HERNI_CAS` a tyto časy uloží do odpovídajících atributů `cas_hrac_1` a `cas_hrac_2` tabulky `HRA`. Implementace této procedury je zobrazena v ukázce kódu 3.3.

Ukázka kódu 3.3: Implementace procedury `KONEC_HRY`

```
1  CREATE OR REPLACE PROCEDURE konec_hry ( hra_id IN INTEGER,
       hrac_id IN INTEGER ) AS
2
3  hra_radek hra%ROWTYPE;
4
5  BEGIN
6
7      SELECT * INTO hra_radek
8      FROM hra
9      WHERE hra.id = hra_id
10         AND rownum = 1;
11
12     UPDATE hra
13     SET
14         cas_hrac_1 = herni_cas ( hra_radek.id, hra_radek.
               hrac_id_1 ),
```

```
15          cas_hrac_2 = herni_cas ( hra_radek.id, hra_radek.
               hrac_id_2 )
16      WHERE id = hra_radek.id;
17
18  END;
19  /
```

3.3.2 AKTUALIZUJ_STAV_HRY

Procedura AKTUALIZUJ_STAV_HRY je také volána při ukončení hry z triggeru VYHODNOT_STAV_HRY (viz kapitola 3.5.1). Tato procedura aktualizuje stav dokončené hry (tj. zda hra skončila vítězstvím začínajícího hráče, prohrou začínajícího hráče, či remízou) a uloží jej do atributu stav_id tabulky HRA. Implementace této procedury je zobrazena v ukázce kódu 3.4.

Ukázka kódu 3.4: Implementace procedury AKTUALIZUJ_STAV_HRY

```
1   CREATE OR REPLACE PROCEDURE aktualizuj_stav_hry ( hra_id IN
       INTEGER, hrac_id IN INTEGER DEFAULT NULL ) AS
2
3   hrac_id_prvni_tah INTEGER;
4   hra_id_prom INTEGER := hra_id;
5
6   BEGIN
7       IF hrac_id IS NULL THEN
8
9           UPDATE hra
10          SET stav_id = 4
11          WHERE id = hra_id_prom;
12
13      ELSE
14
15          SELECT t1.hrac_id INTO hrac_id_prvni_tah
16          FROM tah t1
17      WHERE t1.id =
18        ( SELECT MIN(t2.id) FROM tah t2 WHERE t2.hra_id =
             hra_id_prom );
19
20          IF hrac_id_prvni_tah = hrac_id THEN
21
22              UPDATE hra
23              SET stav_id = 2
24              WHERE id = hra_id_prom;
25
26          ELSE
27
28              UPDATE hra
29              SET stav_id = 3
30              WHERE id = hra_id_prom;
31
32          END IF;
33
34      END IF;
35  END;
36  /
```

3.3.3 REGISTRUJ_HRACE

Tato procedura byla vytvořena k usnadnění registrace hráče. Po jejím zavolání s příslušnými parametry dojde vložení nového záznamu do tabulky `HRAC`. Její implementace je zobrazena v ukázce kódu 3.5.

Ukázka kódu 3.5: Implementace procedury `REGISTRUJ_HRACE`

```
1  CREATE OR REPLACE PROCEDURE registruj_hrace ( prezdivka IN
       NVARCHAR2, id IN INTEGER DEFAULT NULL ) AS
2
3  BEGIN
4
5      IF id IS NULL THEN
6          INSERT INTO hrac
7              ( prezdivka, vyhra_zacinal, vyhra_nezacinal,
                   remiza_zacinal, remiza_nezacinal, prohra_zacinal
                   , prohra_nezacinal )
8          VALUES
9              ( prezdivka, 0, 0, 0, 0, 0, 0 );
10     ELSE
11         INSERT INTO hrac
12             ( id, prezdivka, vyhra_zacinal, vyhra_nezacinal,
                   remiza_zacinal, remiza_nezacinal, prohra_zacinal
                   , prohra_nezacinal )
13         VALUES
14             ( id, prezdivka, 0, 0, 0, 0, 0, 0 );
15     END IF;
16
17     COMMIT;
18
19 END;
20 /
```

3.4 Pohledy

Implementované pohledy odpovídají zadání (viz kapitola 1). V kapitole 3.4.1 je podrobněji popsána implementace pohledu `PAPIR`, v kapitole 3.4.2 je popsána implementace pohledu `VYHRY_ZACINAJICI`.

Skript pro vytvoření všech pohledů je uložen v souboru:

`src\01_create_scripts\04_create_views.sql`

3.4.1 PAPIR

Pohled `PAPIR` reprezentuje čtverečkovaný papír obsahující všechny dosud provedené tahy právě probíhající hry. Pohled nebylo možné realizovat pouze s využitím klauzule `SELECT`, jeho obsah je proto získáván voláním funkce `VYKRESLI_PAPIR` (viz kapitola 3.2.1). Tato funkce vyžaduje jako vstupní parametr `id` hry, pro kterou chceme zobrazit čtverečkovaný papír. Tento parametr je nutné přímo zadat do pohledu `PAPIR`, jelikož v databázovém systému `Oracle` nelze vytvořit pohledy s parametry. Implementace tohoto pohledu je zobrazena v ukázce kódu 3.6.

Ukázka kódu 3.6: Implementace pohledu `PAPIR`

```
CREATE OR REPLACE VIEW papir AS
    SELECT vykresli_papir(101) AS "Papir"
    FROM dual;
```

3.4.2 VYHRY_ZACINAJICI

Tento pohled zobrazí všechny hry, ve kterých zvítězil začínající hráč. Obsahuje parametry hry (rozměry papíru, požadovaná velikost vítězné řady), jména hráčů, kdo začínal (a zvítězil), kdo používal jaké značky, jak dlouho celá hra trvala v sekundách, kolik bylo zahráno tahů. Implementace pohledu `VYHRY_ZACINAJICI` je zobrazena v ukázce kódu 3.7.

Ukázka kódu 3.7: Implementace pohledu `VYHRY_ZACINAJICI`

```
CREATE OR REPLACE VIEW vyhry_zacinajici AS
    SELECT
        h.radek_velikost AS "Pocet␣radku",
        h.sloupec_velikost AS "Pocet␣sloupcu",
        h.symbol_velikost AS "Pocet␣vyhernich␣symbolu",
        hrac1.prezdivka AS "Hrac␣1",
        hrac2.prezdivka AS "Hrac␣2",

        CASE
            WHEN
                (   (
                        SELECT t1.hrac_id
                        FROM tah t1
                        WHERE t1.id =
                            ( SELECT MIN(t2.id) FROM tah t2
                               WHERE t2.hra_id = h.id )
                    ) = hrac1.id
                 )
            THEN
                hrac1.prezdivka
            ELSE
                hrac2.prezdivka
        END AS "Zacinal␣(zvitezil)",

        'O' AS "Hrac␣1␣-␣symbol",
        'X' AS "Hrac␣2␣-␣symbol",
        h.cas_hrac_1 + h.cas_hrac_2 AS "Trvani␣hry",

        (
            SELECT COUNT(t.id)
            FROM tah t
            WHERE t.hra_id = h.id
        ) AS "Pocet␣tahu"

    FROM hra h
        INNER JOIN hrac hrac1 ON hrac1.id = h.hrac_id_1
        INNER JOIN hrac hrac2 ON hrac2.id = h.hrac_id_2
    WHERE h.stav_id = 2;
```

3.5 Triggery

V aplikaci byly implementovány tři triggery, které splňují požadovanou funkcionalitu a automatizaci činností v databázi (viz kapitola 1). Těmito triggery jsou:

- `VYHODNOT_STAV_HRY` – trigger, který hlídá a vyhodnotí stav hry, tj. zda nedospěla do remízy, nebo vítězného stavu jednoho z hráčů (podrobněji v kapitole 3.5.1),
- `ZKONTROLUJ_PARAMETRY_HRY` – hlídá parametry nové hry při vytvoření, jestli splňují daná omezení,
- `ZKONTROLUJ_TAH_HRACE` – hlídá platnost tahu hráče.

Skript pro vytvoření všech triggerů je uložen v souboru:

`src\01_create_scripts\05_create_triggers.sql`

3.5.1 `VYHODNOT_STAV_HRY`

Trigger `VYHODNOT_STAV_HRY` hlídá a vyhodnocuje stav hry, zda nedospěla do konečného stavu.

Tento trigger vyhodnocuje stav hry po každém novém tahu, tj. vložení nového záznamu do tabulky `TAH`. Z nového záznamu je nutné získat souřadnice nového symbolu a vyhodnotit, zda hra nedospěla do vítězství jednoho z hráčů, či remízy. Nicméně, použití *řádkového triggeru* `AFTER INSERT` na tabulku `TAH` není v tomto případě možné. Funkce `VYHRA`, která stav hry vyhodnocuje, kontroluje záznamy v tabulce `TAH`, ale nový záznam ještě není ve chvíli spuštění triggeru v této tabulce obsažen (příkaz `COMMIT` po vložení nového záznamu do tabulky tah je vykonán až po vykonání triggeru). Samotné použití *příkazového triggeru* `AFTER INSERT` na tabulku `TAH` také není možné. Příkazový trigger je sice vykonán až po klauzuli `COMMIT`, nicméně už z jeho podstaty není možné přímo přistupvat k novému záznamu prostřednictvím pseudozáznamu `:new`.

Řešením je použít tzv. `COMPOUND TRIGGER`, v jehož těle lze definovat všechny čtyři typy triggerů (příkazový `BEFORE`, řádkový `BEFORE`, řádkový `AFTER`, příkazový `AFTER`). Trigger `VYHODNOT_STAV_HRY` je právě `COMPOUND TRIGGER` nad tabulkou `TAH` pro operaci `INSERT`.

V tomto triggeru jsou definovány globální proměnné `hra_id_prom` (`id` hry, ve které je nový tah realizován) a `hrac_id_prom` (`id` hráče, který nový tah realizoval). V sekci `AFTER_EACH_ROW` (řádkový trigger) je umožněn přístup k pseudozáznamu `:new` a hodnoty nového řádku jsou v této sekci přiřazeny do definovaných globálních proměnných. V sekci `AFTER STATEMENT` (příkazový trigger) jsou vykonány kontroly stavu hry, kde díky globálním proměnným jsou hodnoty nového záznamu známé. Implementace tohoto triggeru je zobrazena v ukázce kódu 3.8.

Ukázka kódu 3.8: Implementace triggeru `VYHODNOT_STAV_HRY`

```
1  CREATE OR REPLACE TRIGGER vyhodnot_stav_hry
2      FOR INSERT ON tah
3      COMPOUND TRIGGER
4
5      hra_id_prom INTEGER;
6      hrac_id_prom INTEGER;
7
8      AFTER EACH ROW IS
9      BEGIN
10
```

```
11          hra_id_prom := :new.hra_id;
12          hrac_id_prom := :new.hrac_id;
13
14      END AFTER EACH ROW;
15
16      AFTER STATEMENT IS
17      BEGIN
18
19          IF vyhra ( hra_id_prom, hrac_id_prom ) THEN
20              konec_hry ( hra_id_prom, hrac_id_prom );
21              aktualizuj_stav_hry ( hra_id_prom, hrac_id_prom );
22              statistiky ( hra_id_prom );
23          END IF;
24
25          IF remiza ( hra_id_prom ) THEN
26              konec_hry ( hra_id_prom, hrac_id_prom );
27              aktualizuj_stav_hry ( hra_id_prom );
28              statistiky ( hra_id_prom );
29          END IF;
30
31      END AFTER STATEMENT;
32
33
34  END vyhodnot_stav_hry;
35  /
```

4 Nasazení aplikace

Databázová aplikace byla realizována a testována pro prostředí databáze `Oracle` verze `19c` a vyšší.

4.1 Struktura souborů aplikace

Kořenový adresář aplikace obsahuje:

- adresář `doc` – dokumentace aplikace (vč. jejích zdrojových souborů),
- adresář `model` – datový model databáze (vč. souborů k editaci),
- adresář `src` – zdrojové soubory aplikace v jazyce `PL/SQL`,
- soubor `README.md` – obsahuje základní informace o aplikaci (např. jak oskriptovat databázi).

4.1.1 Adresář `src`

Zdrojové soubory aplikace jsou v adresáři `src` rozděleny do podadresářů:

- `01_create_scripts` – skripty k založení všech potřebných databázových objektů (tabulky, funkce, procedury, pohledy, triggery),
- `02_init_data_scripts` – konfigurační data (naplnění tabulek `OMEZENI` a `STAV`),
- `03_scenario_scripts` – testovací scénáře.

4.2 Skriptování databáze

Při skriptování databáze je nutné spouštět skripty v následujícím pořadí:

1. adresář `src\01_create_scripts\`:
   (a) `01_create_tables.sql` – vytvoří tabulky a sekvence,
   (b) `02_create_functions.sql` – vytvoří databázové funkce,
   (c) `03_create_procedures.sql` – vytvoří databázové procedury,
   (d) `04_create_views.sql` – vytvoří databázové pohledy,
   (e) `05_create_triggers.sql` – vytvoří triggery v databázi,
2. adresář `src\02_init_data_scripts\`:
   (a) `01_insert_init_data` – naplní databázi konfiguračními daty (tabulky `OMEZENI` a `STAV`).

Po dokončení běhu všech výše uvedených skriptů jsou v databázi založeny všechny potřebné objekty a konfigurační data, v tuto chvíli je možné databázovou aplikaci používat.

5 Testovací scénáře

Testovací scénáře jsou uloženy v adresáři `src\03_scenario_scripts\` v podadresářích:

- `01_wrong_parameters` – obsahuje testovací scénáře pro zadání špatných parametrů hry a tahu,
- `02_win_first` – obsahuje testovací scénáře pro výhru začínajícího hráče,
- `03_lose_first` – obsahuje testovací scénáře pro prohru začínajícího hráce,
- `04_tie` – obsahuje testovací scénáře pro remízu,
- `05_win_directions` – obsahuje testovací scénáře, zda hra dospěla do vítězného stavu v jednotlivých směrech.

6 Závěr

Cílem této práce bylo navrhnout a vytvořit relační databázovou aplikaci pro hraní známé strategické deskové hry Piškvorky. V databázové aplikaci jsem implementoval všechny databázové objekty popsané v zadání (kapitola 1). Také jsem vytvořil testovací scénáře (viz kapitola 5), které automaticky otestují funkčnost aplikace, včetně chybových stavů.

Při implementaci jsem se setkal se dvěma výraznějšími problémy. Prvním problémem byla implementace pohledu `PAPIR`, který nebylo možné realizovat pouze s využitím klauzule `SELECT`, a proto jsem musel pro vykreslení papíru vytvořit pomocnou funkci. Druhým problémem bylo chování řádkového `AFTER INSERT` triggeru, místo něhož jsem nakonec použil `COMPOUND TRIGGER`.

Semestrální práce z mého pohledu splňuje požadavky zadání.