	Bazy Danych laboratorium	**Laboratorium BD7**

Zagadnienie: Zapoznanie się z SQL Developer Data Modeler w celu tworzenia logicznego i relacyjnego modelu bazy danych

SQL Developer Data Modeler jest narzędziem Oracle umożliwiającym tworzenie schematów logicznych, relacyjnych i generowanie skryptów umożliwiających implementację opracowanych modeli na różne wersje serwerów Oracle i nie tylko.

Oprogramowanie oraz dokumentacja znajdują się na stronie:

https://www.oracle.com/database/technologies/appdev/datamodeler.html

I. Etapy modelowania bazy danych

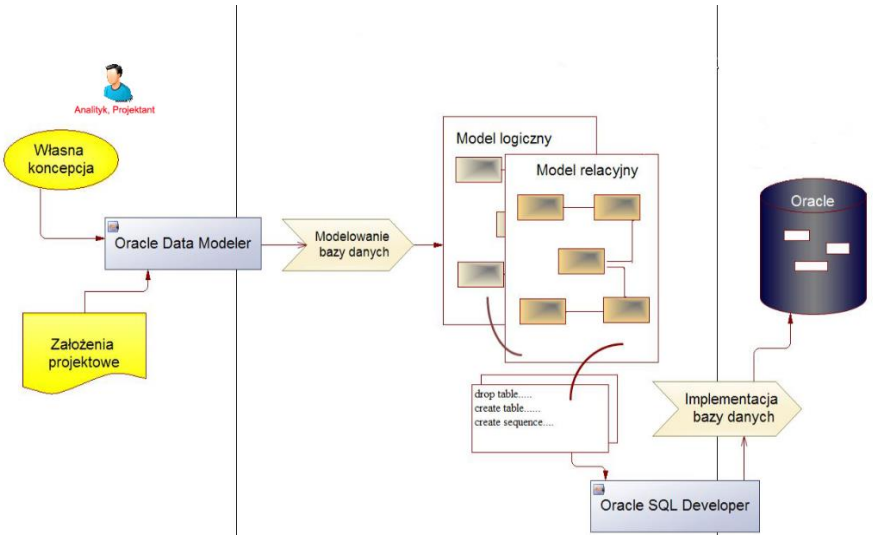
Modelowanie bazy danych składa się z kilku etapów:

1. Modelowanie pewnego fragmentu rzeczywistości czyli przedstawienie tej rzeczywistości w postaci modelu logicznego składającego się z encji i związków między encjami. Każda encja posiada swoją nazwę oraz strukturę, na którą składają się atrybuty opisujące dany obiekt i typy tych atrybutów. Co najmniej jeden z atrybutów musi być wyróżnikiem, przy pomocy którego można jednoznacznie identyfikować dany egzemplarz encji.

2. Na podstawie modelu logicznego tworzony jest model relacyjny, w którym na podstawie encji powstają tabele, a związki zostają transformowane na relacje. Tabele składają się z kolumn i ich typów. Pojawiają się klucze główne i klucze obce.

3. Model relacyjny poddawany jest analizie i ewentualnym modyfikacjom zmierzającym do przekształcenia go do wymaganych postaci normalnych (najczęściej do 3NF).

4. Na podstawie znormalizowanego modelu relacyjnego tworzony jest model fizyczny w języku SQL umożliwiający implementację na konkretnym serwerze bazodanowym.

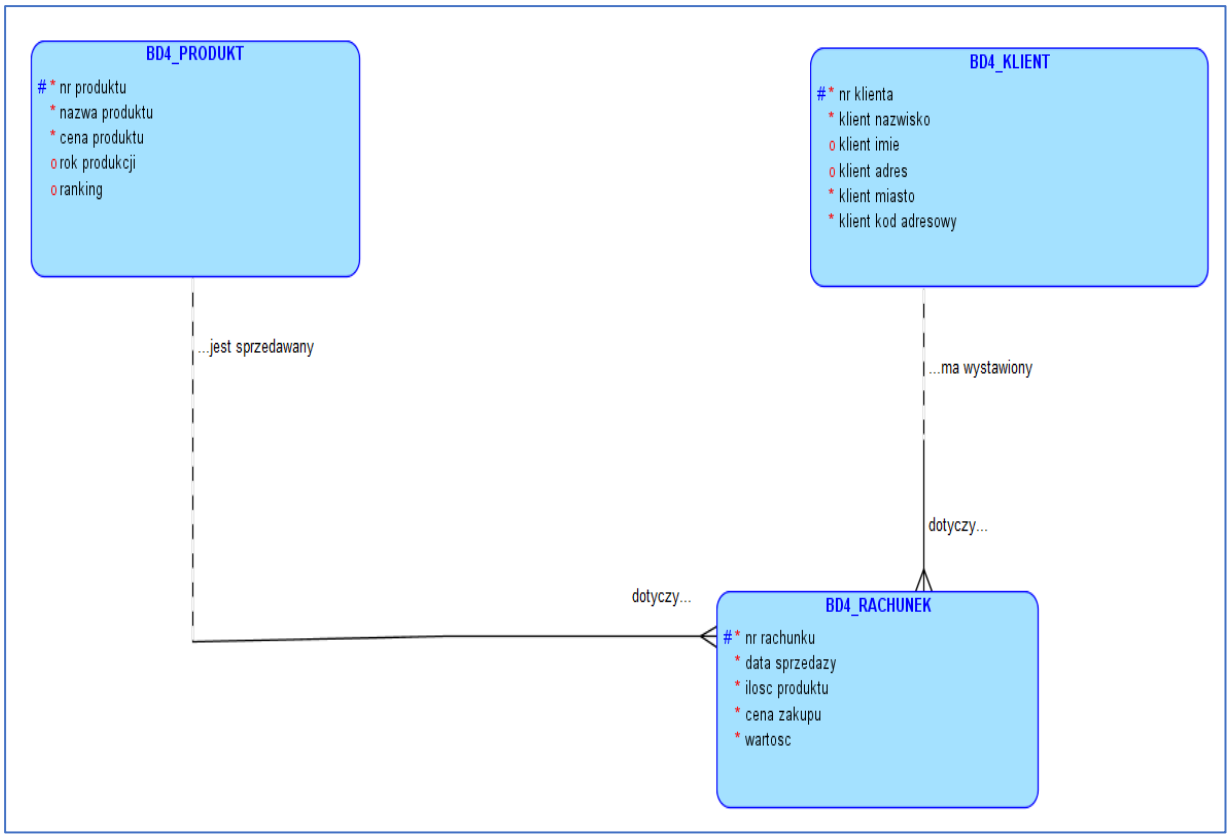


II. Projektowanie modelu bazodanowego z dwiema relacjami 1:N (one to many)

Należy opracować model logiczny *Rachunek* w oparciu o poniższe założenia.

Firma sprzedaje swoje produkty zarejestrowanym klientom i wystawia za te transakcje rachunki. Każda transakcja dotyczy jednego produktu czyli na rachunku widnieje tylko jedna pozycja sprzedanego towaru. W modelu *Rachunek* należy przechowywać informacje dotyczące sprzedaży produktów klientom według modelu: Klient, Produkt, Rachunek.

Finalna postać modelu logicznego przedstawiona jest na poniższym rysunku:



1. Zdefiniować trzy obiekty logiczne (*entity*) według poniższych założeń:

Entity BD4_PRODUKT:

Nazwa atrybutu	Typ atrybutu	Informacje dodatkowe
nr produktu	numeric (5)	Unikalny identyfikator
nazwa produktu	varchar (20)	Musi być wypełnione
cena produktu	numeric (8,2)[1]	Musi być wypełnione
rok produkcji	numeric (4)	
ranking	numeric (2)	Ranking produktów:[2] 1 – produkt bardzo zły, 10 - produkt bardzo dobry

[1] Numeric(8,2) oznacza liczbę zawierającą 6 cyfr przed kropką dziesiętną i 2 cyfry po kropce. Należy ustawić Precision na 8 i Scale na 2.
[2] Znaczenie tego atrybutu należy opisać w komentarzu do definicji *entity* (Comments in RDBMS).

Entity BD4_KLIENT:

Nazwa atrybutu	Typ atrybutu	Informacje dodatkowe
nr klienta	numeric (4)	Unikalny identyfikator
klient nazwisko	varchar (30)	Musi być wypełnione
klient imie	varchar (15)	
klient adres	varchar (50)	
klient miasto	varchar (20)	Musi być wypełnione
klient kod adresowy	varchar (6)	Musi być wypełnione

Entity BD4_RACHUNEK:

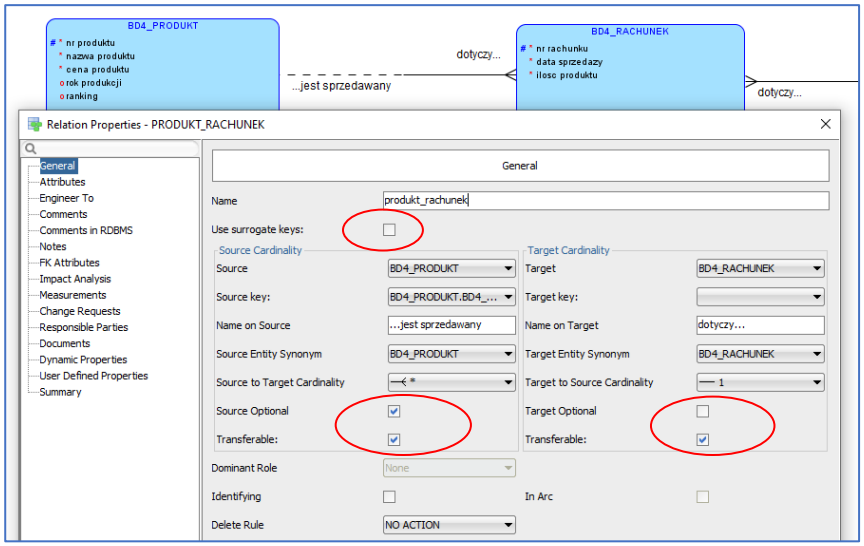
Nazwa atrybutu	Typ atrybutu	Informacje dodatkowe
nr rachunku	numeric (4)	Unikalny identyfikator
data sprzedazy	date	Musi być wypełnione
ilosc produktu	numeric (3)	Musi być wypełnione
cena zakupu	numeric(8,2)	Musi być wypełnione
wartosc	numeric(8,2)	Musi być wypełnione

2. Zapisać powstały fragment modelu logicznego pod nazwą *Rachunek* (*save as…*), a następnie poprzez funkcję *View/Logical Diagram Notation* zapoznać się z dwiema oferowanymi przez *Data Modeler* konwencjami prezentacji diagramów logicznych (notacje Barkera i Bachmana).

3. Zdefiniować związki między obiektami przyjmując następujące założenia:

   - Związek *produkt_rachunek* (jeden_do_wielu): każdy produkt może być wiele razy sprzedawany klientom, ale również mogą być produkty ani razu nie sprzedane.

   W tym celu z paska narzędzi wybrać odpowiednią ikonę relacji 1:N, a następnie połączyć encje BD4_PRODUKT i BD4_RACHUNEK przeciągając myszką od encji BD4_PRODUKT.
   Pojawi się formularz właściwości relacji, w którym można dokonać niezbędnych modyfikacji poprzez zmianę nazwy relacji, słownych interpretacji relacji (Name of Source i Name of Target)[3] oraz siły relacji (Cardinality), np.:



[3] Aby te określenia pojawiły się na diagramie należy na pulpicie głównym przy pomocy PPM (prawy przycisk myszy) wybrać opcję *Show / Labels*.

Pozostawienie niewybranej opcji Target Optional oznacza, że w transakcji sprzedaży musi być określony produkt czyli w strukturze przyszłej tabeli BD4_RACHUNEK pozycja nr_produktu musi być określona. Gdyby ta opcja była zaznaczona oznaczałoby to, że kolumna nr_produktu będzie miała właściwość null czyli transakcja może dotyczyć nieokreślonego produktu.

- Związek *klient_rachunek* (jeden_do_wielu): każdy fakt sprzedaży dotyczy jednego określonego klienta, klient może wiele razy kupować produkty, ale są klienci, którzy nie dokonali żadnej transakcji.

Dla tego związku ustawić w taki sam sposób jego parametry jak dla produkt_rachunek.

W panelu Browser odnaleźć definicje związków między encjami (Relations) i zapoznać się ze strategiami postępowania w przypadku kasowania wpisów (Delete Rule). Dla związku klient_rachunek ustawić strategię na restrykcyjną (RESTRICT), co oznacza, że nie będzie można usunąć wiersza z tabeli nadrzędnej (BD4_KLIENT), jeśli w tabeli podrzędnej (BD4_RACHUNEK) znajdują się odpowiadające mu wiersze. Dla relacji produkt_rachunek ustawić strategię na CASCADE, co oznacza, że w przypadku kasowania wiersza w tabeli nadrzędnej (BD4_PRODUKT), w tabeli podrzędnej (BD4_RACHUNEK) zostaną skasowane automatycznie wszystkie wiersze będące w relacji z kasowanym wierszem nadrzędnym.[4]

4. Przy pomocy funkcji *Engineer To Relational Model* (PPM na poziomie *Logical Model* lub ikona >>) wygenerować relacyjny model (przycisk *Engineer*). Szczególną uwagę zwrócić na tabelę *BD4_RACHUNEK*, w której to zostały dołożone dwie kolumny stanowiące klucze obce. Jeśli kolumny te mają nazwy *BD4_PRODUKT_nr_produktu* i *BD4_KLIENT_nr_klienta* należy wykonać czynności opisane w przypisie[5] i ponownie wygenerować model relacyjny. W panelu *Browser* znaleźć w folderze *Relational Models* wygenerowany *Relational_1* i poprzez PPM na tej nazwie i *Properties* zmienić nazwę modelu relacyjnego na *rachunek* oraz *RDBMS Type* na *Oracle Database 12cR2*. Poprzez *Save* zapamiętać projekt.

5. *Data Modeler* umożliwia tworzenie obiektów fizycznego modelu danych (na poziomie konkretnego serwera bazodanowego). Poprzez panel *Browser* należy rozwinąć model relacyjny *rachunek* i na poziomie *Physical Models* przy pomocy PPM wybrać konkretny serwer (np. Oracle Database 12cR2). Odszukać w tej strukturze znane obiekty bazodanowe.

6. Utworzyć sekwencję *seq_klient*, która będzie zaczynała się od 10 i dawała przyrost kolejnych wartości o 1, sekwencję *seq_produkt* zaczynającą się od 20 z interwałem 1 oraz sekwencję *seq_rachunek* generującą wartości od 1 z inkrementacją 1. Dla wszystkich ustawić *Disable Cache* na *Yes.*

7. Tym samym sposobem można tworzyć obiekty programowe (PL/SQL), takie jak wyzwalacze, procedury i funkcje definiując tylko ich specyfikację i pozostawiając kodowanie na późniejszy termin.
   Odnaleźć folder *Stored Procedures* w *Physical Models* i zdefiniować prototyp procedury o nazwie *pr_insert_produkt*, przy pomocy której wprowadzany będzie nowy produkt do tabeli *BD4_PRODUKT*. Po wpisaniu nazwy procedury w polu *Name* i zatwierdzeniu formularza pojawi się panel *SQL Developer* wraz z niekompletnym prototypem procedury. Należy go doprowadzić do poniższej postaci:

[4] Inną opcją określającą zasady kasowania jest *SET NULL*. Taka definicja powoduje, że kasowanie wiersza nadrzędnego powoduje ustawienie w wierszach podrzędnych na pozycji klucza obcego wartości *null*. Ma to sens tylko w przypadku ustawienia opcji *Target Optional* na *Yes*

[5] Przed wygenerowaniem modelu relacyjnego można zmienić ustawienia nazewnictwa poprzez przejście przy pomocy prawego przycisku myszy do właściwości diagramów (w panelu Browser -> Design *rachunek* -> PPM -> Properties -> Settings -> Naming Standard -> Templates). Wartość w polu Foreign Key zmienić na *{parent}_FK*, a w Column Foreign Key na *{ref column}*.

```
CREATE OR REPLACE PROCEDURE pr_insert_produkt
                            (v_nr_produktu number,
                             v_nazwa_produktu varchar2,
                             v_cena_produktu number,
                             v_rok_produkcji number default null,
                             v_ranking number default 5) AS
BEGIN
  NULL;
        /*
          Procedura wstawia nowy produkt do tabeli BD4_PRODUKT.
          Należy sprawdzać, czy podany numer produktu już istnieje.
          Rok produkcji i ranking nie muszą być wypełnione.
        */
END;
```

i zamknąć ten panel "z krzyżyka" potwierdzając zmiany.

8. Przejść do ustawień *Tools / Preferences... / Data Modeler / DDL* i wyłączyć opcję *Include Default Settings in DDL* oraz *Include Logging in DDL.*

9. Przy pomocy funkcji *File /Export /DDL File /Generate* wygenerować dwa skrypty dla serwera Oracle. Pierwszy o nazwie *rachunek_create.ddl* (zakładka *Create Selection*) zawierający zdania SQL tworzące model bazy danych oraz drugi o nazwie *rachunek_drop.ddl* (zakładka *Drop Selection* - należy wybrać zaprojektowane obiekty: tabele. indeksy, klucze obce, sekwencje *)* zawierający dodatkowo zdania SQL usuwające tabele i powiązania między nimi. W obu przypadkach zawrzeć w skrypcie komentarze wprowadzone w modelu logicznym (*Include Comments*).

   W trakcie generowania skryptów może pojawić się informacja o błędzie w projekcie. Odnaleźć w skrypcie szczegóły informacji i dokonać odpowiednich poprawek. Na przykład nazwa wygenerowanej relacji między tabelami może być za długa. Zmienić w modelu nazwy obu relacji, a następnie powtórnie wygenerować skrypt lub ustawić opcje nazewnictwa na podstawie przypisu z poprzedniej strony.

   W skrypcie tworzącym bazę danych zwrócić uwagę na różnice w definicjach kluczy obcych wynikające z przyjętej logiki modelu:

```
ALTER TABLE bd4_rachunek
    ADD CONSTRAINT bd4_klient_fk FOREIGN KEY ( nr_klienta )
        REFERENCES bd4_klient ( nr_klienta );

ALTER TABLE bd4_rachunek
    ADD CONSTRAINT bd4_produkt_fk FOREIGN KEY ( nr_produktu )
        REFERENCES bd4_produkt ( nr_produktu )
            ON DELETE CASCADE;
```

   Relacja bd4_klient_fk jest restrykcyjna, co oznacza, że nie jest możliwe skasowanie klienta z ewidencji (obiekt nadrzędny) jeśli istnieje wpis o zakupie przez niego co najmniej jednego produktu. Relacja bd4_produkt_fk umożliwia automatyczne skasowanie wszystkich transakcji dotyczących kasowanego produktu z katalogu (obiekt nadrzędny).

   Skrypt rachunek_drop.ddl zawiera w sobie zarówno zdania kasujące obiekty modelu jak i zdania tworzące te obiekty. Dla sprawności wdrożeniowej lepiej jest, aby te dwie funkcjonalności rozdzielić na poziomie skryptów. Dlatego też należy w skrypcie rachunek_drop.ddl pozostawić tylko zdania SQL usuwające obiekty bazodanowe ze schematu.

10. Przy pomocy narzędzia *SQL Developer* lub *SQLPlus* uruchamiać oba wygenerowane skrypty na serwerze oraz wypełnić utworzone tabele danymi wykorzystując dołączony do materiałów skrypt *rachunek_populate.sql*:

    ```
    @c:\temp\rachunek_drop.ddl
    @c:\temp\rachunek_create.ddl
    @c:\temp\rachunek_populate.sql
    ```

    Sprawdzić poprawność implementacji wykonując zdanie:

    ```
    select 'BD4_KLIENT' as "Tabela", count ( * ) "Liczba wierszy" from bd4_klient
    union
    select 'BD4_PRODUKT', count ( * ) from bd4_produkt
    union
    select 'BD4_RACHUNEK', count ( * ) from bd4_rachunek;
    ```

Tabela	Liczba wierszy
BD4_KLIENT	7
BD4_PRODUKT	10
BD4_RACHUNEK	200

11. Przetestować i przeanalizować skuteczność działania więzów integralnościowych zdefiniowanych w modelu poprzez próby realizacji poniższych czynności i obserwując skutki tego działania:

    a. Podjąć próbę usunięcia z tabeli *BD4_KLIENT* dowolnego klienta,

    b. Obliczyć w tabeli BD4_RACHUNEK liczbę transakcji związanych z wybranym produktem, usunąć ten produkt z tabeli BD4_PRODUKT i ponownie dokonać tego samego obliczenia,

    c. Wykonać zdanie SQL *rollback* i ponownie powyższe (pkt. 10) zdanie sprawdzające.

12. Przejść do właściwości *Data Modeler* (*Tools* / *Preferences*), zapoznać się i ewentualnie zmienić niektóre ustawienia środowiska, np.:

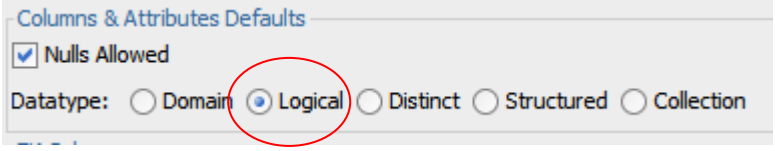
    a. ścieżki dostępu do katalogów roboczych (*Data Modeler*),

    b. standardową notację w modelu logicznym (*Data Modeler* / *Diagram* / *Logical Model*),

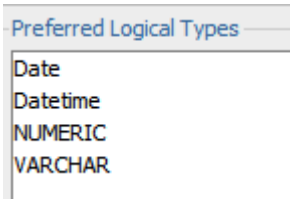
    c. kierunek strzałek na diagramie modelu relacyjnego (*Data Modeler* / *Diagram* / *Relational Model)*,

    d. standardowy RDBMS (*Data Modeler* / *Model)* – *Oracle Database 12cR2*,

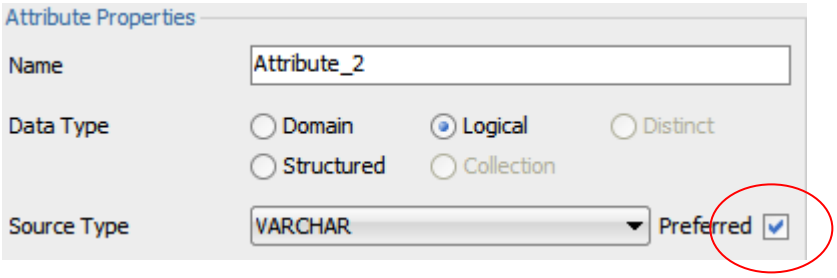
    e. ograniczenie wyboru typów danych przy projektowaniu encji (*Data Modeler* / *Model):*

    ....



Aby powyższe ustawienie stało się aktywne należy w trakcie projektowania encji ustawić opcję *Preferred*:



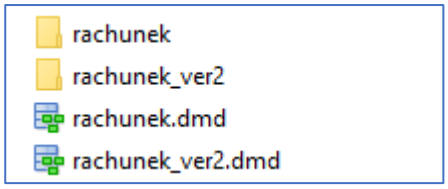
13. Przy pomocy przycisku *Export* (znajdującego się na formularzu *Preferences / Data Modeler)* wyeksportować do własnego folderu opracowane opcje. Mogą one być potem importowane w innej instalacji *Data Modeler* z tego samego miejsca przyciskiem *Import*.

14. Zakończyć pracę z *Data Modeler*.

III. Projektowanie modelu bazodanowego z relacją M:N (many to many)

Należy opracować drugą wersję modelu *rachunek* (wcześniej zaimplementowanego) w oparciu o takie same założenia. Istotną różnicą będzie przedstawienie modelu logicznego w postaci dwóch encji (*KLIENT* i *PRODUKT*) będących ze sobą w związku M:N.

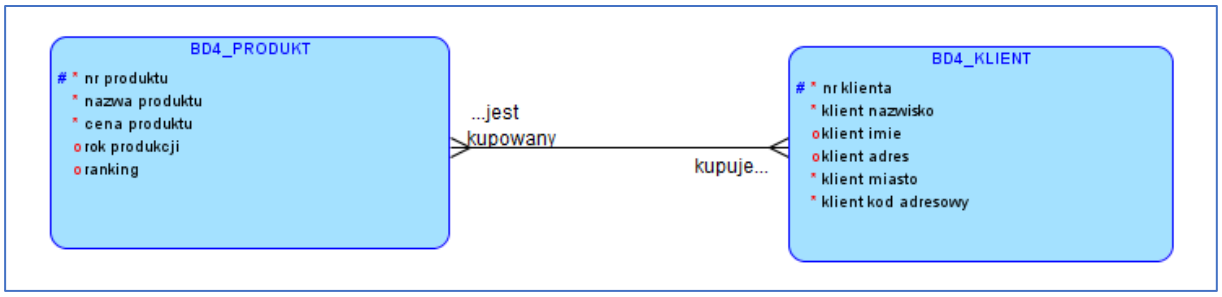
1. W *Data Modeler* otworzyć opracowany logiczny model *rachunek*[6], a następnie zapamiętać go (*save as*...) pod nazwą *rachunek_ver2*.



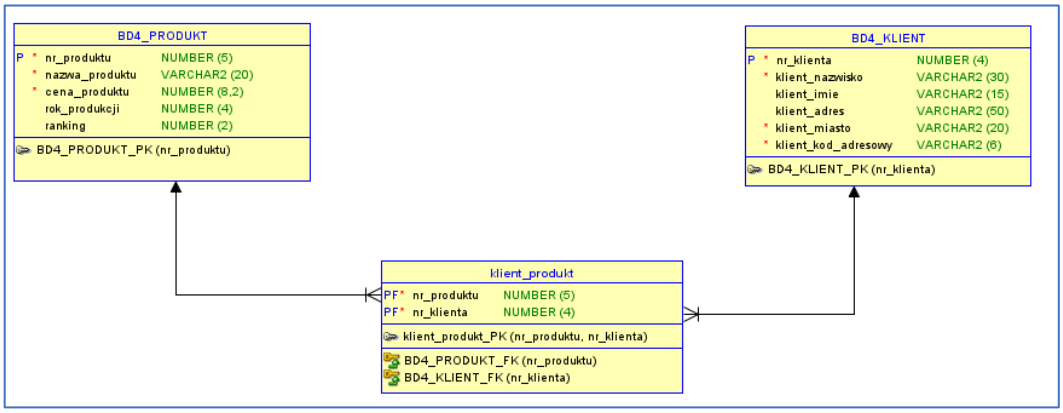
2. W tej wersji modelu logicznego usunąć encję *BD4_RACHUNEK*. Automatycznie zostaną usunięte oba zdefiniowane uprzednio związki między encjami. Natomiast w modelu relacyjnym *rachunek* tej wersji projektu usunąć manualnie występujące tam tabele i zmienić jego nazwę na *rachunek_ver2*. Zapewni to utrzymanie w nim uprzednio zdefiniowanych sekwencji i procedury *pr_insert_produkt.*

3. Między encjami *BD4_PRODUKT* i *BD4_KLIENT* utworzyć związek *wiele do wielu* (M:N) i nazwać go *klient_produkt*. Określić *Name of Source* na "...jest kupowany", a *Name of Target* na "kupuje...". *Source Optional* i *Target Optional* powinny być ustawione na *No* (niezaznaczone), podobnie jak opcja *Use surrogate keys*.

[6] Na formularzu *Select Relational Models* zaznaczyć wybór modelu relacyjnego i fizycznego.

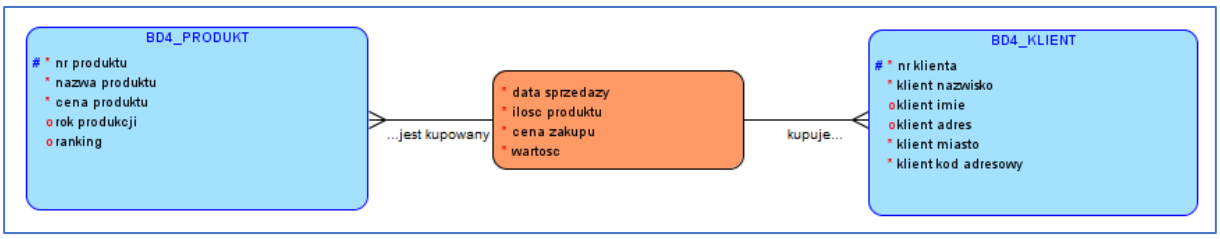


4. Wygenerować model relacyjny. Powstała trzecia tabela, w której klucz główny jest kluczem złożonym zbudowanym na kolumnach będących kluczami głównymi w pozostałych tabelach:

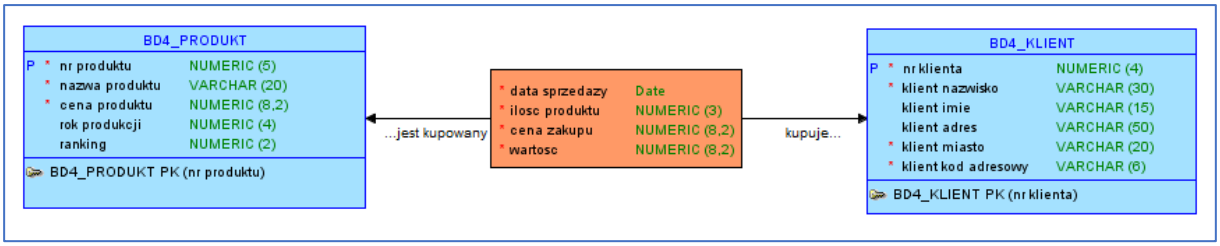


5. Powrócić do edycji modelu logicznego. Otworzyć właściwości związku *klient_produkt* (np. *Browser / Logical Model / Relations*) i zdefiniować dodatkowe atrybuty tego związku (przejść do *Attributes*):
   - *data sprzedazy* typu Date obowiązkowy,
   - *ilosc produktu* typu Numeric (3) obowiązkowy,
   - *cena zakupu* typu Numeric (8,2) obowiązkowy,
   - *wartosc* typu Numeric (8,2) obowiązkowy

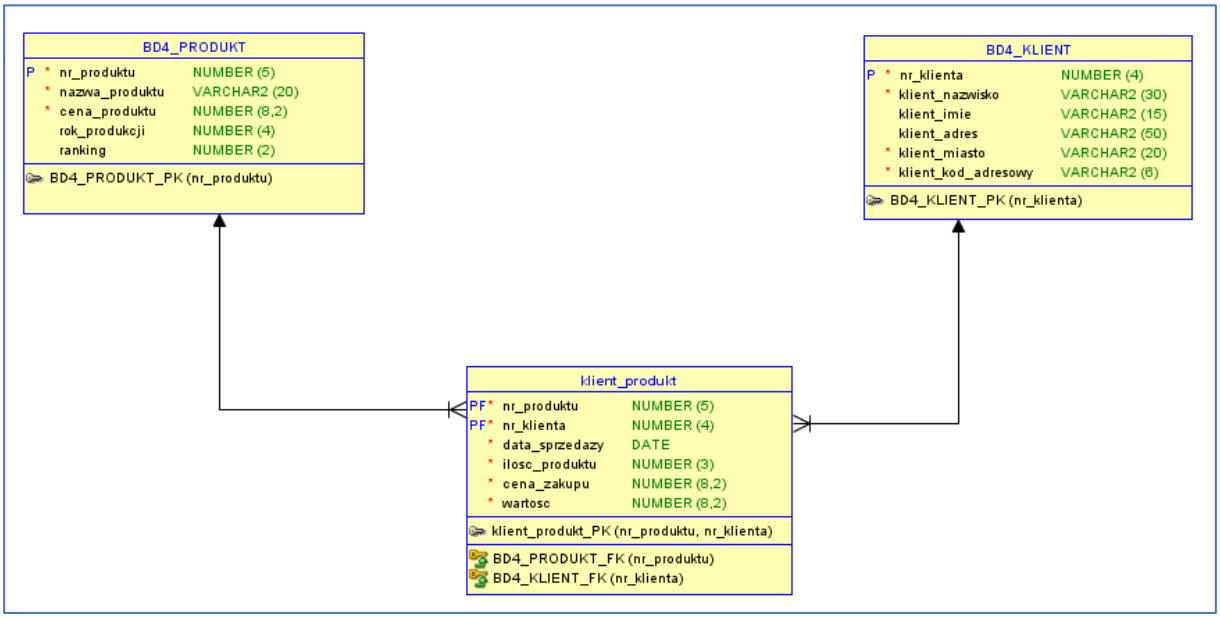
   , zatwierdzić te definicje, a następnie w dowolnym miejscu panelu przy pomocy PPM wybrać *Show / Relationship Attributes*.



Zmienić notację diagramu z notacji Barkera na Bachmana (w panelu PPM i *Notation*).

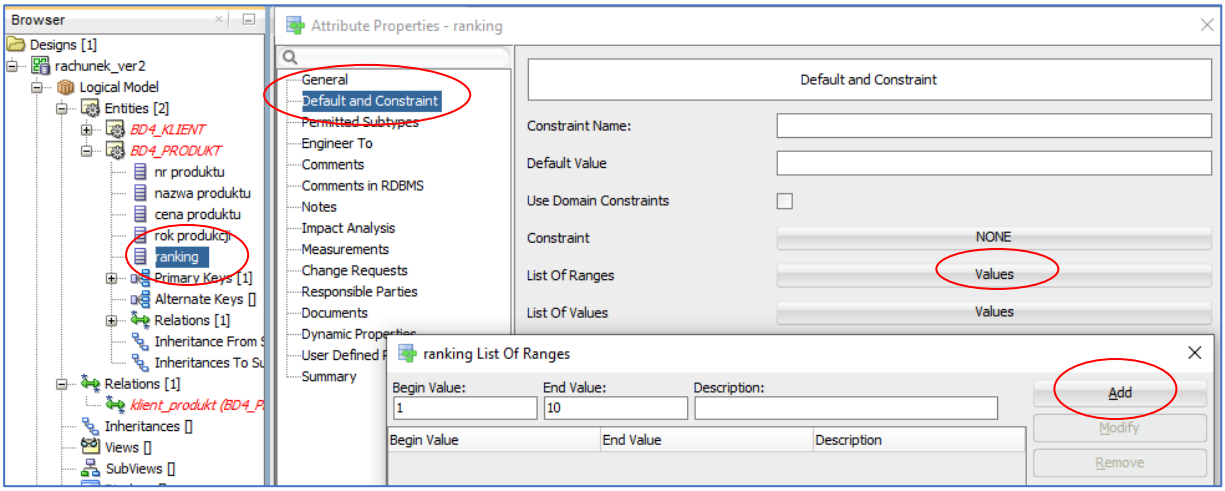


6. Powtórnie wygenerować model relacyjny:

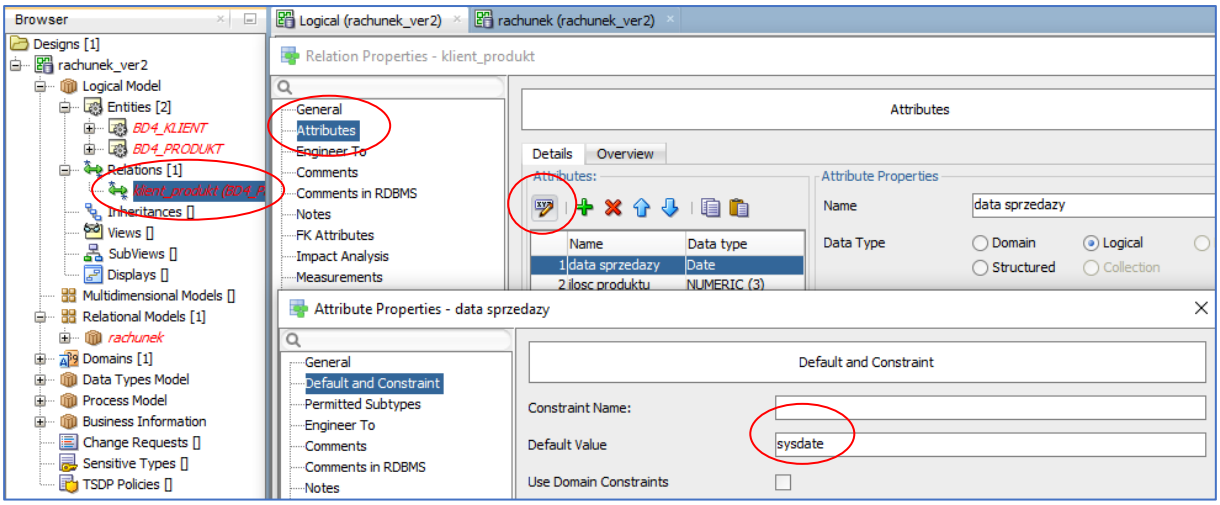


Dodatkowe definicje związku między encjami zostały uwzględnione w tabeli pośredniczącej (intersekcji).

7. Powrócić do edycji modelu logicznego. Wybrać właściwości atrybutu *ranking* w encji *BD4_PRODUKT* i ustawić zakres dopuszczalnych wartości na przedział [1..10]:



8. Podobnie dla atrybutu *data sprzedazy* ustawić bieżącą datę jako domyślną:

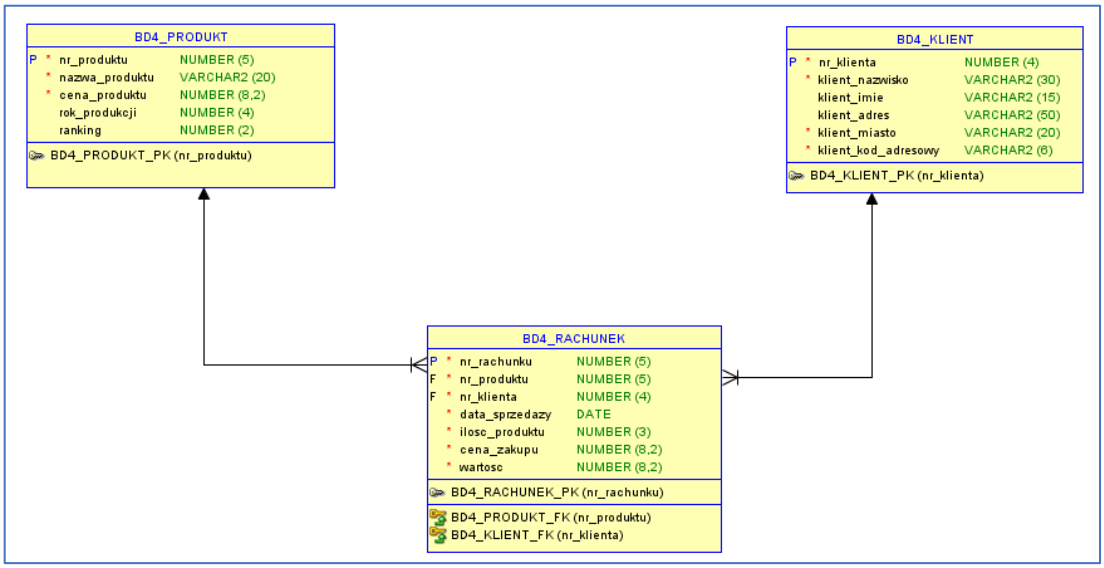


I wygenerować model relacyjny.

9. Utworzony model relacyjny zawiera tabelę pośrednią (*klient_produkt*), której kluczem głównym jest para numer klienta i numer produktu. Jest to wada tego modelu, gdyż oznacza to, że dany klient może kupić dany produkt tylko jeden raz, a to nie jest zgodne z rzeczywistością. Należy ten model zmodyfikować wprowadzając do tabeli *klient_produkt* nowy klucz główny (*nr_rachunku* NUMBER (5)) umieszczając go dodatkowo na pierwszej pozycji w tabeli oraz usuwając tę właściwość z kolumn *nr_klienta* i *nr_produktu* (te modyfikacje można wykonać otwierając właściwości tabeli).

10. Na koniec zmienić nazwę tabeli na *BD4_RACHUNEK* (na formularzu *General*) oraz nazwę definicji klucza głównego (na formularzu *Primary Key*) z *klient_produkt_PK* na *BD4_RACHUNEK_PK*.

Ostateczna postać modelu relacyjnego powinna wyglądać tak:



Należy zaznaczyć, że po tych modyfikacjach nie jest gwarantowane ponowne prawidłowe generowanie modelu relacyjnego na podstawie modelu logicznego. Innymi słowy, jeśli

dokonalibyśmy modyfikacji w modelu logicznym i ponownie wygenerowali model relacyjny to należałoby ponownie dokonywać zmian w wygenerowanym modelu relacyjnym, które zostały zrobione w punkcie 9 i 10.

11. Wygenerować nową wersję skryptu DDL (*rachunek_create_ver2.ddl)* i ocenić jego zgodność z czynnościami wykonanymi na projektowanym modelu.

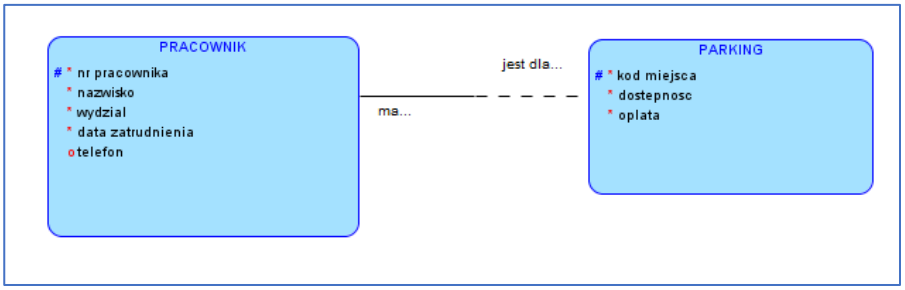
12. Wprowadzić przykładowe dane do tabel (zmodyfikowanym odpowiednio skryptem *rachunek_populate_ver2.sql* na podstawie *rachunek_populate.sql*). Sprawdzić skuteczność działania opracowanych restrykcyjnych strategii kasowania wierszy w tabelach.

IV. Projektowanie modelu bazodanowego z relacją 1:1 (one to one)

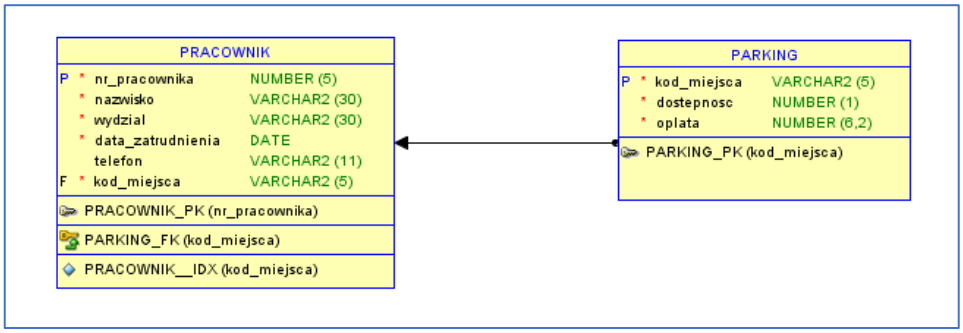
Związek 1:1 definiuje zasadę, że w modelu logicznym pojedynczy egzemplarz jednej encji jest w związku z co najwyżej jednym egzemplarzem drugiej encji. Jak łatwo zauważyć, trudno w takim przypadku mówić o hierarchii obiektów (nadrzędny – podrzędny).

Załóżmy, że firma dla swoich pracowników organizuje na swoim terenie miejsca parkingowe. Zakłada się, że każdy pracownik obligatoryjnie ma przydzielone jedno i tylko jedno takie miejsce.

Model logiczny mógłby wyglądać tak (budowa związku 1:1 od *PARKING* do *PRACOWNIK*)[7]:



, a odpowiadający mu model relacyjny:



W tak wygenerowanym modelu relacyjnym należy zwrócić uwagę na trzy aspekty:

1. W tabeli *PRACOWNIK* został umieszczony klucz obcy pochodzący z tabeli *PARKING*. Właściwość istnienia jest ustawiona na *obowiązkowość* (*), co przekłada się na definicję kolumny:

   .....
   kod_miejsca VARCHAR2(5) ***NOT NULL***
   .....

   A to oznacza, że każdy pracownik, z urzędu, ma przydzielone miejsce parkingowe.
2. W tabeli *PRACOWNIK* automatycznie został wygenerowany indeks *PRACOWNIK_IDX* oparty na kolumnie klucza obcego *kod_miejsca*:

[7] Należy usunąć możliwość użycia klucza sztucznego (opcja *Use surrogate keys* ma być niewybrana).

*CREATE **UNIQUE** INDEX pracownik_idx ON*
PRACOWNIK (kod_miejsca ASC);

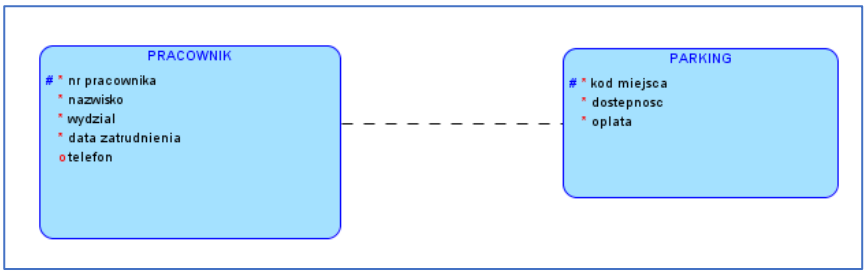
Jest to indeks unikalny, co oznacza, że w kolumnie *kod_miejsca* tabeli *PRACOWNIK* nie może wystąpić więcej niż raz ta sama wartość. Spełnia to założenie przydziału każdemu pracownikowi jednego miejsca parkingowego.

3. Gdyby zmienić założenie na opcjonalność przydziału miejsca parkingowego czyli pracownik nie musi, ale może posiadać takowe, wystarczy w modelu logicznym ustawić w definicji związku *Target Optional* na Yes oraz jako *Dominant Role* wybrać *PARKING*. Wtedy w wygenerowanym na nowo modelu relacyjnym kolumna *kod_miejsca* otrzyma atrybut *null*:

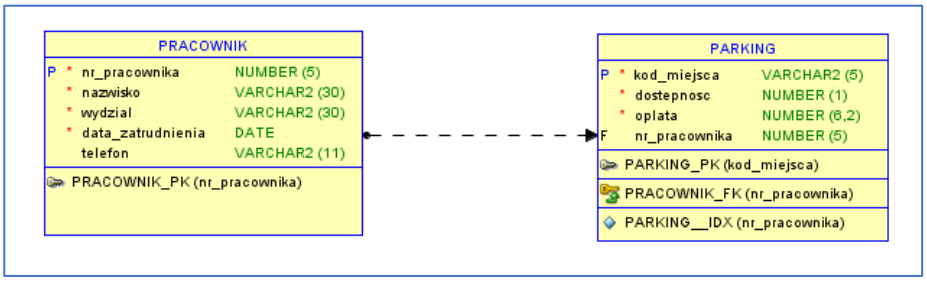
.....
kod_miejsca VARCHAR2(5)
.....

Dopuszczenie istnienia wartości *null* w kolumnie *kod_miejsca* nie ma wpływu na działanie unikalnego indeksu *pracownik_idx*, gdyż wystąpienia *null*, w takim przypadku, nie podlegają analizie na powtarzalność wartości.

Warto rozważyć sytuację odwrotną do omawianej do tej pory budując związek od *PRACOWNIK* do *PARKING*:



, a wygenerowany model relacyjny tak:



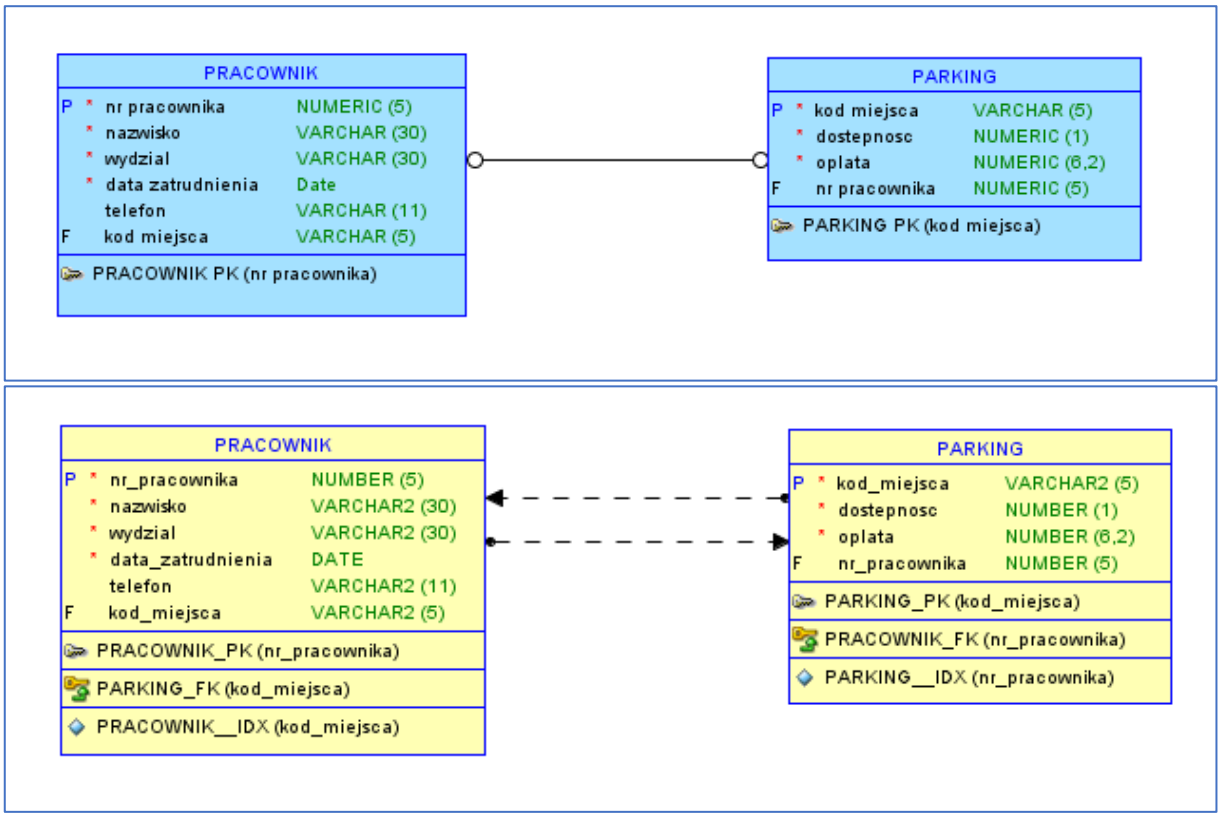
Nadal spełnione są założenia, że pracownik może (ale nie musi) mieć przydzielone co najwyżej jedno miejsce parkingowe (istnieje unikalny indeks *PARKING_IDX*).
Istotna różnica może się pojawić w przypadku chęci skasowania pracownika z ewidencji.
W pierwszym wariancie modelu można uznać nadrzędność tabeli *PARKING* nad tabelą *PRACOWNIK*, gdyż klucz obcy jest usadowiony w tej drugiej. Skasowanie wiersza w tabeli podrzędnej (*PRACOWNIK*) nie zakłóca spójności bazy danych i może być bez przeszkód wykonane.

W wariancie drugim mamy sytuację odwrotną. Tabela *PRACOWNIK* jest tabelą nadrzędną, gdyż klucz obcy znajduje się w tabeli *PARKING*. Kasowanie wiersza nadrzędnego nie jest takie oczywiste. Jeśli trzeba skasować pracownika, który ma przydzielone miejsce parkingowe to system zarządzania bazą danych do tego nie dopuści z racji zachowania spójności bazy danych, jeśli relacja między tabelami będzie zdefiniowana jako restrykcyjna (RESTRICT). Nielogicznym

biznesowo byłoby najpierw skasowanie odpowiedniego wiersza w tabeli *PARKING*, a następnie pozostającego z nim w relacji pracownika w tabeli *PRACOWNIK* z racji utraty danych o miejscu parkingowym (samym w sobie).
Najlepszym, w takiej sytuacji, rozwiązaniem jest modyfikacja relacji między tabelami poprzez zmianę zasady dotyczącej kasowania wiersza nadrzędnego *Delete Rule* z RESTRICT na Set NULL. Można to zmienić w modelu relacyjnym w definicji klucza obcego *PRACOWNIK_FK* zmieniając strategię *Delete Rule* na SET NULL. W wygenerowanym skrypcie implementującym model objawi się to zdaniem:

```
ALTER TABLE PARKING
    ADD CONSTRAINT PRACOWNIK_FK FOREIGN KEY ( nr_pracownika )
    REFERENCES PRACOWNIK ( nr_pracownika )
     ON DELETE SET NULL;
```

Teraz, kasując pracownika w tabeli *PRACOWNIK* równocześnie w odpowiadającym mu wierszu tabeli *PARKING* w kolumnie *nr_pracownika* zostanie wstawiony *null*, ale wiersz pozostanie.
Inną metodyką modelowania związków *one_to_one* jest wprowadzanie równoczesne do obu tabel kluczy obcych naprzemiennie. Należy w modelu logicznym w definicji związku między encjami zlikwidować dominującą rolę jednej z nich (*Dominant Role* ustawić na None) oraz *Source* i *Target Optional* na Yes.



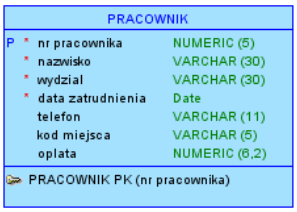
Każdy pracownik może, ale nie musi, mieć przydzielone co najwyżej jedno miejsce parkingowe, a każde miejsce parkingowe może, ale nie musi, być przydzielone co najwyżej jednemu pracownikowi.

W takim modelu problemem jest zagwarantowanie dwustronności relacji między wierszami obu tabel. Jeśli w tabeli *PRACOWNIK* Abacki ma przydzielone miejsce parkingowe A-101, to jak zapewnić, aby w tabeli *PARKING* miejsce parkingowe A-101 było przydzielone Abackiemu?
Potrzebne są rozwiązania programowe (wyzwalacze, procedury), których kod może zapewnić poprawność danych.

Uwagi końcowe:

1. W modelach relacyjnych najczęściej występują związki *one_to_many* oraz rzadziej *one_to_one*. Natomiast nie mogą występować związki *many_to_many*, gdyż takiego związku nie można implementować. Przetwarza się je na dwa związki *one_to_many.*

2. Związek *one_to_one* można zrekonfigurować w ten sposób, że wszystkie atrybuty obu encji umieścić w jednej:



   Wybór odpowiedniej techniki zależny jest od konkretnego obszaru modelowania i nie sprawdza się w każdym przypadku. Tabela powstała na podstawie powyższej encji nie jest dobrym przykładem, gdyż nie jest tabelą znormalizowaną (*oplata* nie jest zależna funkcyjnie od *nr_pracownika* tylko od *kod_miejsca*).

3. Obszarami zastosowania związków *one_to_one* mogą być dane częściowo wrażliwe. Tabela danych pracowników zawierająca dane typowe, jak również dane wrażliwe może być zdekomponowana na dwie tabele. Tabela z danymi wrażliwymi podlegająca specjalnej ochronie może być szyfrowana.

4. Innym przykładem może być sytuacja, gdy tabela zawiera dane bardzo często przetwarzane i rzadko przetwarzane, ale pozostające ze sobą w związku *one_to_one*. Z analizy wydajności działania bazy można wysnuć wniosek, że opłaca się podzielić tę tabelę na dwie oddzielne, aby zmniejszyć wielkość transferów z/do pamięci dyskowej, a przez to poprawić wydajność bazy.

V. Informacje dodatkowe i uzupełniające

Do tej pory omówione zostały podstawowe zasady tworzenia modelu logicznego i relacyjnego przy użyciu związków i relacji typu *one_to_many* i *many_to_many* w najczęściej spotykanych zastosowaniach.
Poniżej zostaną przedstawione inne zasady modelowania wynikające z ogólnej teorii modelowania baz danych oraz z możliwości Oracle Data Modeler.

Jak zaznaczono wcześniej model logiczny składa się z encji i związków zachodzących między encjami. Każdy związek posiada trzy cechy: stopień związku, typ asocjacji (kardynalność) oraz cechę istnienia (klasę przynależności).

Stopień związku określa liczbę encji połączonych związkiem. Wyróżnia się związki unarne (łączące encję samą ze sobą), binarne (łączące dwie encje) oraz n-arne (łączące n encji).

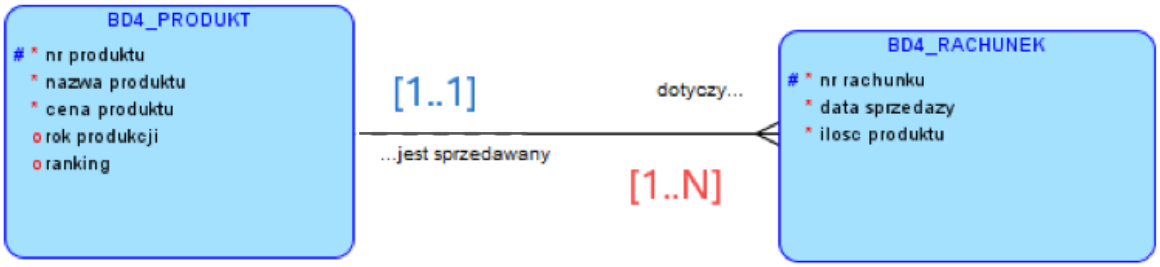
Typ asocjacji określa ile egzemplarzy jednej encji może wchodzić w związek z jednym egzemplarzem drugiej encji. Wyróżnia się typy: *one_to_one*, *one_to many* i *many_to_many*.

Cecha istnienia określa czy dany związek jest opcjonalny czy obowiązkowy.

W tym materiale omówione zostaną tylko związki binarne wszystkich typów asocjacji i obu cech istnienia.

Związek one to many

Związek 1:N definiuje zasadę, że w modelu logicznym pojedynczy egzemplarz encji nadrzędnej jest w związku z pewną liczbą egzemplarzy encji podrzędnej. Czyli jeden produkt **może** występować na wielu rachunkach, ale każdy rachunek **musi** dotyczyć jednego produktu. W wielu notacjach modelowania fakt ten zobrazowuje się na diagramach przy pomocy pary [1..N] lub podobnej.

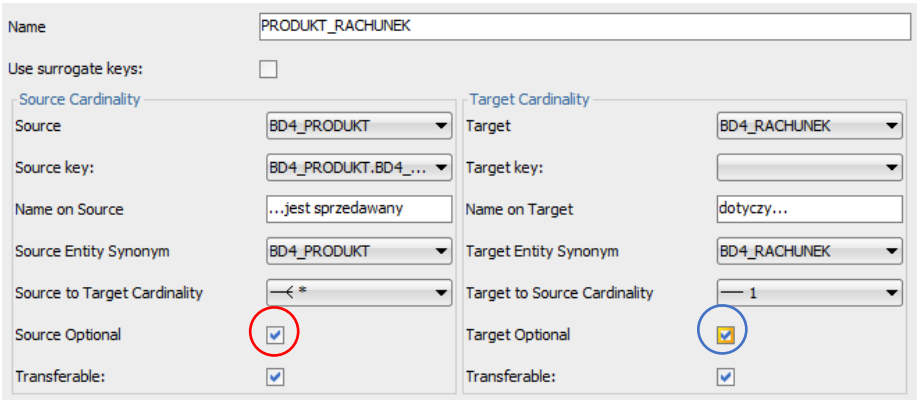


Oznaczenie [1..N] określa, że każdy produkt musi mieć co najmniej jedno wystąpienie w encji *rachunek*, a oznaczenie [1..1] określa, że z każdym egzemplarzem encji *RACHUNEK* związany jest jeden i tylko jeden produkt. Innymi słowy – nie może egzystować w ewidencji produktów produkt, który nie został kupiony, z racji definicji [1..N] oraz każdy rachunek musi zawierać jeden produkt ([1..1]).
W omawianym przykładzie jest to sprzeczne z logiką biznesową, gdyż katalog produktów może zawierać produkty nie kupione do tej pory. Aby być w zgodzie z taką logiką należy zastosować kolejną właściwość związków między encjami, a mianowicie istnienie (*opcjonalność / obowiązkowość)* związku.

Obowiązkowość wyrażana jest zapisem [1..N], co oznacza, że każdy egzemplarz encji nadrzędnej musi być w związku z co najmniej jednym egzemplarzem encji podrzędnej. Natomiast opcjonalność oznaczana jest jako [0..N] czyli egzemplarz encji nadrzędnej nie musi być w związku z żadnym egzemplarzem encji podrzędnej (mogą istnieć w ewidencji produktów te, które nie były jeszcze kupione ani razu). I taka definicja jest odpowiednia dla omawianego przykładu.

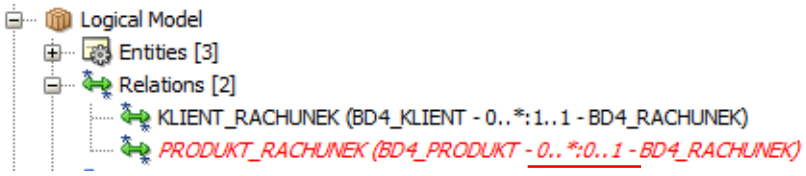
Podobnie oznaczenie [1..1] określa, że każdy egzemplarz encji podrzędnej musi być związany z dokładnie jednym egzemplarzem encji nadrzędnej, a oznaczenie [0..1] – że ten związek może zachodzić, ale nie musi. Innymi słowy – dopuszcza się istnienie transakcji bez określonego produktu.

W środowisku Oracle Data Modeler tę właściwość definiuje się przy pomocy ustawienia *Source Optional* i *Target Optional*:



Zmieniając wartości obu ustawień można uzyskać cztery możliwe warianty istnienia związku między dwiema encjami. Zmiany te można obserwować rozwijając w *Browser Logical Model / Relations*.

Dla powyższych ustawień związek *PRODUKT_RACHUNEK* jest zdefiniowany następująco:



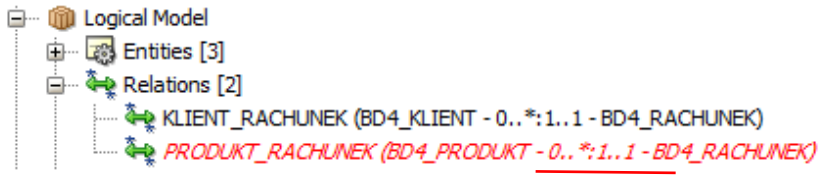
, co odpowiada notacji: [0..N] i [0..1].

Po wygenerowaniu modelu relacyjnego oraz skryptu ddl, jego fragment zawiera poniższą definicję:

```
CREATE TABLE bd4_rachunek (
    nr_rachunku      NUMBER(4) NOT NULL,
    data_sprzedazy   DATE NOT NULL,
    ilosc_produktu   NUMBER(3) NOT NULL,
    nr_produktu      NUMBER(5),
    nr_klienta       NUMBER(4) NOT NULL
);
```

, czyli dopuszcza się nieokreśloność produktu w transakcji zakupu.

Kombinacja [0..N] i [1..1] czyli *Source Optional*: wybrana, a *Target Optional*: nie wybrana daje inny efekt:



, zmiana jest widoczna na diagramie modelu relacyjnego oraz w wygenerowanym skrypcie:

```
CREATE TABLE bd4_rachunek (
    nr_rachunku      NUMBER(4) NOT NULL,
    data_sprzedazy   DATE NOT NULL,
    ilosc_produktu   NUMBER(3) NOT NULL,
    nr_produktu      NUMBER(5) NOT NULL,
    nr_klienta       NUMBER(4) NOT NULL
);
```

, czyli każda transakcja zakupu musi dotyczyć określonego produktu.

W podobny sposób można analizować dwie pozostałe kombinacje ustawień istnienia związku czyli [1..N] i [0..1] oraz [1..N] oraz [1..1].

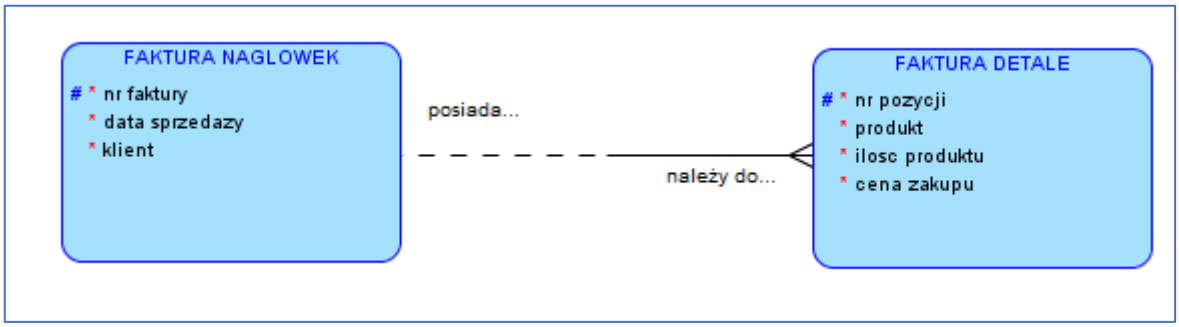
Narzędzia służące modelowaniu baz danych nie mają środków umożliwiających zdefiniowanie związku: „każdy element encji nadrzędnej musi być w związku z co najmniej jednym egzemplarzem encji podrzędnej” ([1..N]), czyli w katalogu produktów są tylko takie produkty, które

chociaż raz były kupione. Tak postawiony problem rozwiązuje się metodami programowymi na poziomie serwera bazodanowego (wyzwalacze, procedury) lub na poziomie aplikacji.

Słaba encja (weak entity)

Słaba encja to szczególny typ encji, w której identyfikator (atrybut kluczowy) składa się (przynajmniej częściowo) z atrybutów kluczowych innych encji.

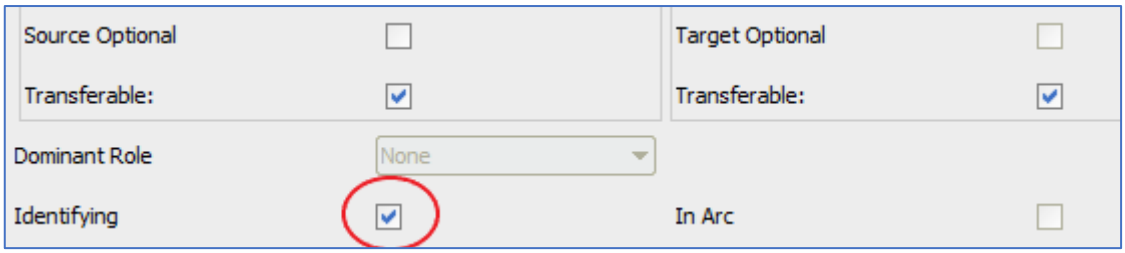
Przykładem niech będzie fragment modelu logicznego fakturowania wielopozycyjnego:



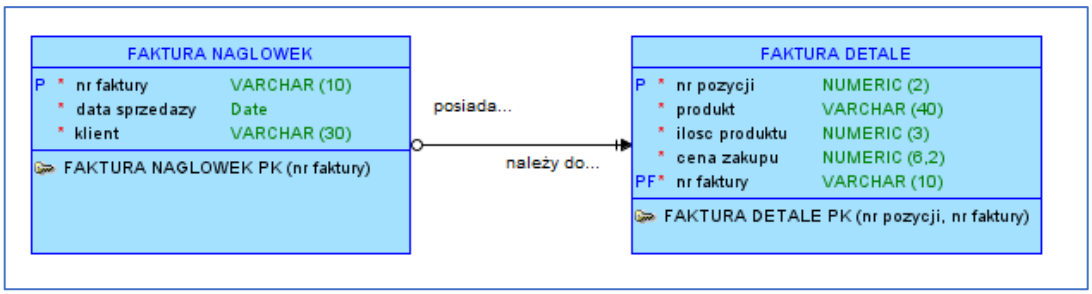
Faktura składa się z nagłówka oraz pozycji faktury. Jeden egzemplarz encji *FAKTURA_NAGLOWEK* jest w związku 1:N z wieloma egzemplarzami encji *FAKTURA_DETALE*.

Obie encje mają swoje własne identyfikatory. *#nr faktury* jednoznacznie identyfikuje każdy egzemplarz encji *FAKTURA_NAGLOWEK*, ale *#nr pozycji* w odniesieniu do encji *FAKTURA_DETALE* – nie, gdyż dla każdej faktury numeracja pozycji zaczyna się od wartości 1 (np. nr faktury 100/AB/19 ma pozycje 1,2 i 3, a nr 101/AB/19 – 1,2,3 i 4). Dopiero para tych identyfikatorów w sposób jednoznaczny określa egzemplarz encji *FAKTURA_DETALE*.

Aby to osiągnąć w Oracle Data Modeler należy, określając związek między encjami, wybrać odpowiednią definicję związku (*1:N Relation Identifying*).
We właściwościach zostanie to uwzględnione poprzez zaznaczenie opcji *Identifying*:



Zmieniając wygląd diagramu tego modelu logicznego z notacji *Barkera* na *Bachmana*:



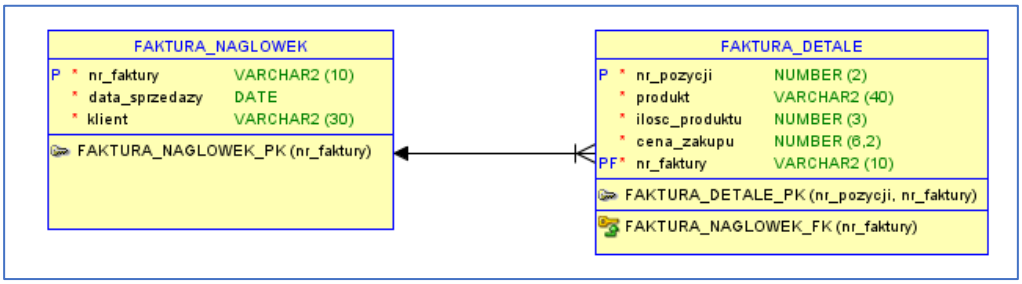
można zauważyć, że w prawidłowy sposób została zaprojektowana encja detali faktury.

Identyfikator jest złożony i częściowo zależny od identyfikatora encji nagłówka faktury.

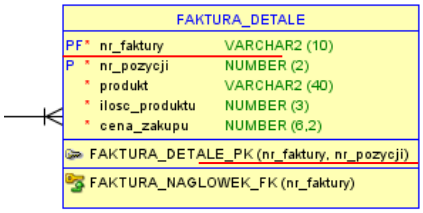
I to jest istotą encji słabych.

Chcąc wygenerować prawidłowo model relacyjny należy w obu encjach poprzez *Entity Properties / General* usunąć możliwość tworzenia klucza sztucznego (*Create Surrogate Key*), jeśli z jakiś względów ta możliwość jest zaznaczona.

Wygenerowany model może wyglądać tak:



Dla porządku można zarówno w modelu logicznym (notacja *Bachmana*) i w modelu relacyjnym przenieść *nr_faktury* w detalach faktury na pierwszą pozycję (zarówno w strukturze tabeli jak i w definicji klucza głównego).



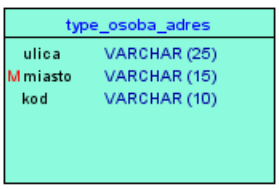
VI. Projektowanie modelu uwzględniającego dziedziczenie obiektów i typy strukturalne

Opracować projekt logiczny OSOBA w oparciu o poniższe założenia:

Model logiczny powinien zawierać trzy encje: OSOBA, NAUCZYCIEL i STUDENT, przy czym encja OSOBA ma pełnić rolę obiektu nadrzędnego, co oznacza, że dwie pozostałe encje mają, oprócz posiadania własnych atrybutów, dziedziczyć jej atrybuty. Dodatkowo należy zdefiniować własny typ strukturalny określający adres zamieszkania. Każda z osób (zarówno nauczyciele jak i studenci) może posiadać kilka kont w funkcjonujących systemach informatycznych.

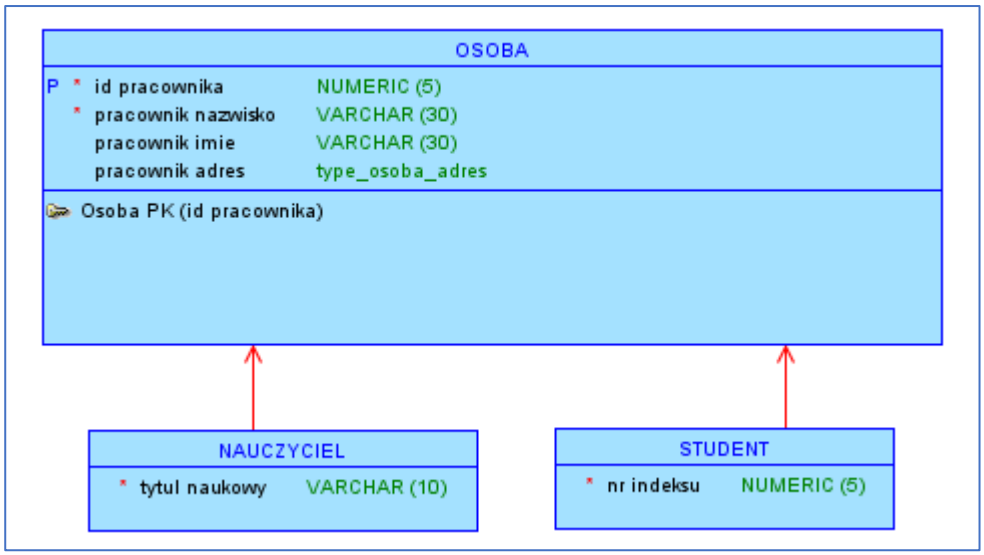
1. W celu utworzenia własnego typu strukturalnego należy w lewym panelu odnaleźć folder *Data Types Model* i przy pomocy prawego przycisku myszy wybrać *Show*, a następnie z głównego menu odpowiednią ikonę na pasku narzędziowym (*New Structured Type*) i kliknąć w panelu roboczym.

2. Nadać nazwę tworzonemu obiektowi: *type_osoba_adres* i przejść do definiowania atrybutów według poniższego rysunku analogicznie do definiowania atrybutów encji:



Litera M (Mandatory) przy atrybucie oznacza, że musi być on zawsze określony.

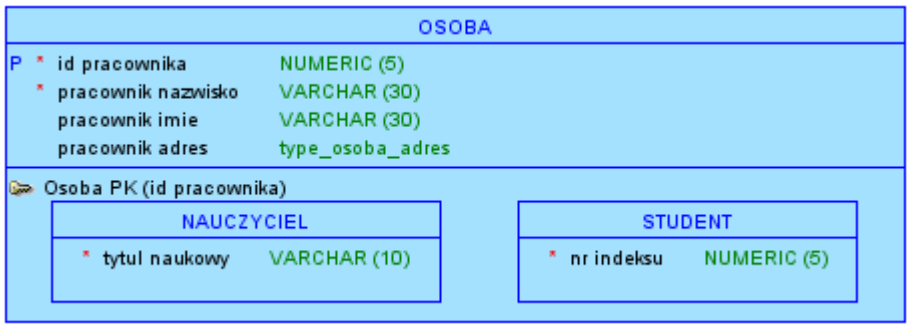
3. Zdefiniować w standardowy sposób trzy encje OSOBA, NAUCZYCIEL i STUDENT według poniższych diagramów:



Dla atrybutu *pracownik adres* wybrać dla *Data Type* opcję *Structured,* a następnie rozwinąć Source *Type* i wybrać zdefiniowany typ.

4. Przystąpić do modyfikacji encji NAUCZYCIEL i STUDENT polegającej na określeniu obiektu, z którego tworzone encje będą dziedziczyły atrybuty. Realizuje się to poprzez wybór obiektu nadrzędnego w polu *Super Type* we właściwościach modyfikowanej encji (*Properties / General*).

5. Sposób prezentacji dziedziczenia obiektów można zmienić. W tym celu należy w panelu roboczym kliknąć prawym przyciskiem myszy poza zdefiniowanymi obiektami i wybrać funkcję *Notation / Box-In-Box Presentation*, a następnie poprzez zabiegi modyfikujące wygląd doprowadzić do ostatecznej postaci:

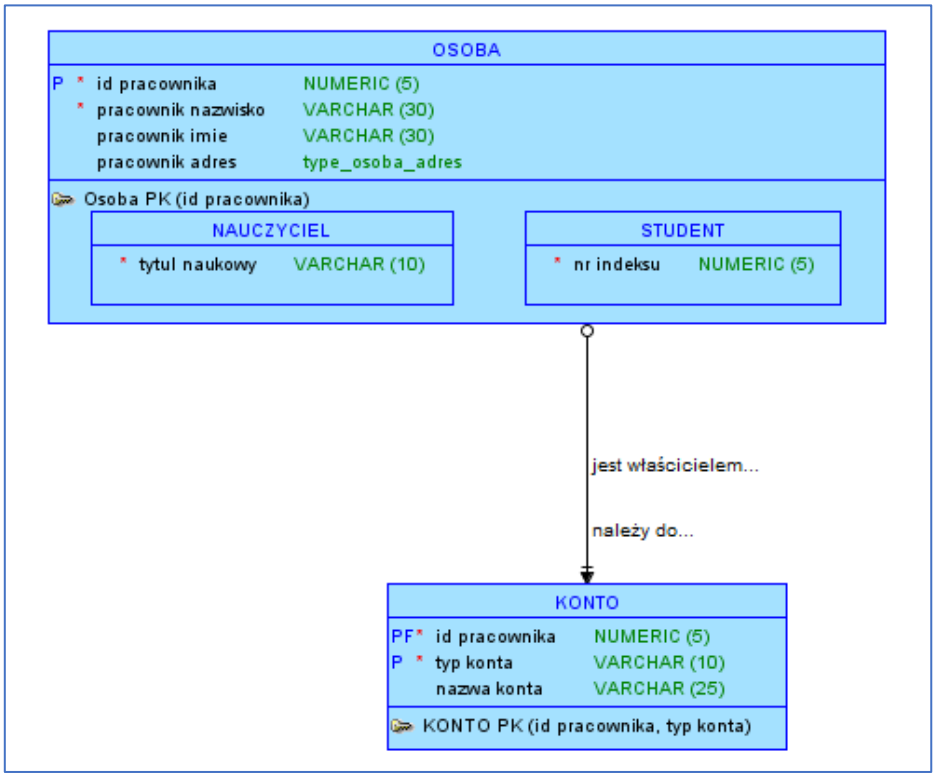


Sprawdzić sposób prezentacji w obu typach notacji (Bachman i Barker).

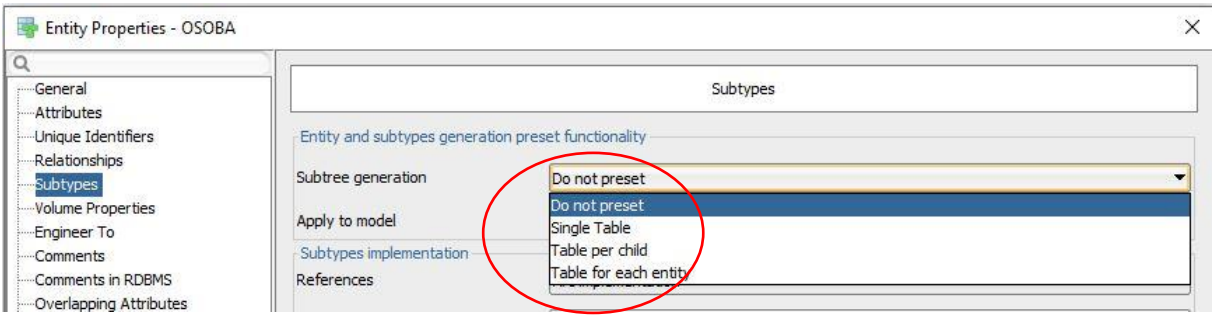
6. Zamodelować encję KONTO odzwierciedlającą przydzielanie konta komputerowego pracownikom uczelni (nauczycielom i studentom), przy czym dopuszcza się posiadanie kilku kont przez jedną osobę[8]. Encja posiada dwa atrybuty: *typ_konta varchar(10)* jako unikalny identyfikator i nieobowiązkowy *nazwa_konta varchar(25)*. Dodatkowo należy

[8] W definicji encji KONTO nie należy definiować atrybutu *id_pracownika*. Po określeniu związku (1:N) między encjami identyfikator pracownika stanie się kluczem obcym. Należy wtedy dodatkowo zdefiniować go jako część klucza głównego i umieścić na odpowiedniej pozycji. Można to zrobić ustawiając w definicji relacji *osoba_konto* opcję *Identifying* oraz kasując opcję *Use surrogate keys*.

zdefiniować dopuszczalne wartości atrybutu *typ_konta*, np.: *mail, oracle, teams* i *extranet*. Można to zrealizować we właściwościach atrybutu (*Attribute Properties*) i w zakładce *Default and Constraints* nazywając te więzy np. *typ_konta_ch* oraz edytując pozycję *List Of Values* wprowadzając założone nazwy typów kont.



7. Przed przystąpieniem do generowania relacyjnych modeli należy zwrócić uwagę na ustawienie opcji odpowiedzialnej za sposób implementacji tabel OSOBA, NAUCZYCIEL i STUDENT. W tym celu dla encji OSOBA z głównych właściwości encji odczytać wartości w polu *Subtree generation* (*General / Subtypes*).



Istnieje kilka sposobów transformacji takiego modelu logicznego.
Poniżej zostanie zaprezentowany sposób oparty o jedną tabelę (*Single Table*) zawierającą implementację atrybutów wszystkich encji (OSOBA, NAUCZYCIEL i STUDENT).

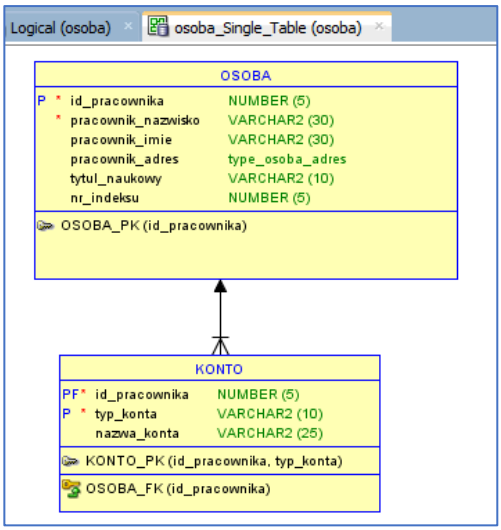
8. Ustawić we właściwościach encji OSOBA:

   - *Subtree generation: Single Table,*
   - *Apply to model: Relational_1,*
   - *References: Identifying,*

a następnie uruchomić *Engineer to Relational Model*. Zmienić nazwę wygenerowanego modelu relacyjnego na *osoba_Single_Table*.

9. W tabeli KONTO zmienić nazwę kolumny osoba_id_pracownika na id_pracownika (w razie potrzeby) .

10. Przeanalizować powstały model relacyjny. Zwrócić uwagę na połączenie struktur encji OSOBA, NAUCZYCIEL i STUDENT w jedną tabelę.



Dodatkowo warto zauważyć, że obowiązkowość atrybutów występujących w encjach NAUCZYCIEL i STUDENT (*nr_indeksu* i *tytul_naukowy*) została usunięta, co ma swoje logiczne uzasadnienie. W każdym wierszu tabeli OSOBA musi wystąpić *null* albo w kolumnie *nr_indeksu* albo *tytul_naukowy*. Może to być jedna z metod rozróżniania pracowników uczelni. Można to zrealizować wybierając dla tabeli OSOBA (w modelu relacyjnym) poprzez *Properties* / *Table Level Constraints* i tworząc własną regułę walidacji:

Name: OSOBA_CH
Validation Rule:

(nr_indeksu is null and tytul_naukowy is not null)
or
(nr_indeksu is not null and tytul_naukowy is null)

Po wygenerowaniu skryptu można zauważyć sposób implementacji tej reguły w postaci zdania SQL

```
alter table osoba
        add constraint osoba_ch check
                ((nr_indeksu is null and tytul_naukowy is not null)
        or
                (nr_indeksu is not null and tytul_naukowy is null));
```

umożliwiającą kontrolę poprawności wprowadzanych i modyfikowanych danych.

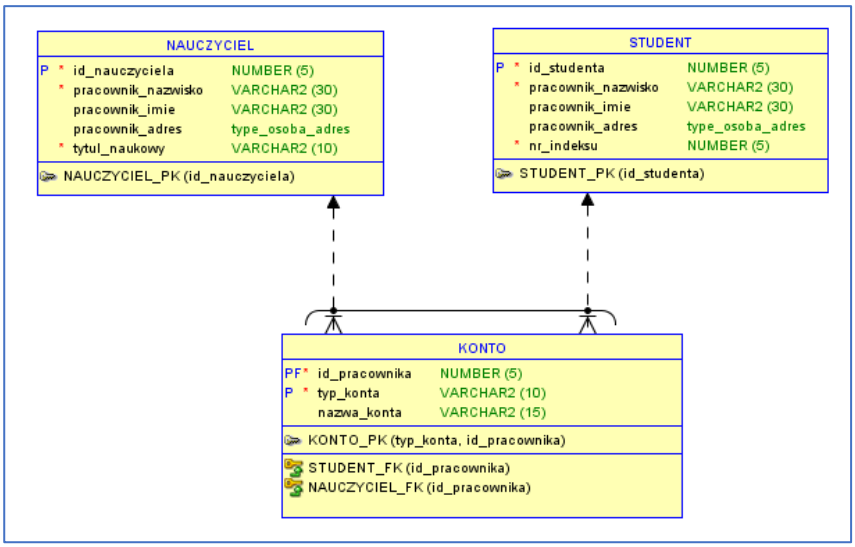
Skuteczność tego wariantu można przetestować realizując zdania SQL zawarte w dołączonym do materiałów skrypcie *osoba_inherit.sql*.

Inne warianty generowania modelu relacyjnego umieszczone w Oracle Data Modeler nie będą prezentowane w tym materiale.

Projektant lub programista może innymi metodami gwarantować rozróżnialność danej osoby w tworzonej aplikacji, na przykład wprowadzając w tabeli OSOBA status osoby w postaci kolumny z dopuszczalnymi wartościami określającymi nauczyciela i studenta.

Innym sposobem może być utworzenie dwóch oddzielnych tabel NAUCZYCIEL i STUDENT (bez tabeli OSOBA), dla których generowanie kolejnych wartości klucza głównego odbywa się poprzez jedną wspólną sekwencję, czyli definicje kolumn kluczy głównych mogą wyglądać tak:

*id_nauczyciela numer(5) default **seq_osoba.**nextval primary key*
*id_studenta numer(5) default **seq_osoba**.nextval primary key*



W tym przypadku właściwość klucza obcego w tabeli KONTO (***F** id_pracownika*) nie może być zagwarantowana na poziomie SQL w prosty sposób[9], ale można to zrobić wspomagając się językiem PL/SQL, na przykład opracowując procedurę *pr_insert_konto*:

```
create or replace procedure pr_insert_konto
                    (v_nr_pracownika numer,
                     v_typ_konta varchar2 default 'mail',
                     v_nazwa_konta varchar2) as
begin
          null;
end;
```

, która będzie dokonywała niezbędnych walidacji, aby programowo zapewnić spójność bazy danych czyli nie dopuścić do założenia konta nauczycielowi lub studentowi nie figurującemu w odpowiedniej ewidencji.

[9] Przykład rozwiązania można znaleźć pod adresem http://db-oriented.com/2016/03/11/implementing-arc-relationships-with-virtual-columns/ lub https://stackoverflow.com/questions/61624669/how-do-you-recreate-the-arc-relationship-in-the-oracle-database-modeler-in-oracl