

Złączenia – pułapki i nowe możliwości

Jarosław Gramacki, Artur Gramacki

*Uniwersytet Zielonogórski
Instytut Informatyki i Elektroniki
ul. Podgórna 50, 65-246, Zielona Góra
e-mail: j.gramacki@iie.uz.zgora.pl, a.gramacki@iie.uz.zgora.pl*

Abstrakt

Celem niniejszego artykułu jest omówienie dostępnych w systemie Oracle rodzajów połączeń (ang. *joins*) tabel relacyjnych. Zagadnienie łączenia tabel jest tak stare, jak zasady, na których opiera się model relacyjny. Niemniej liczba dostępnych w systemie Oracle siedmiu (!) różnych terminów posiadających w swej nazwie frazę „join” (*Equijoins, Self Joins, Cartesian Products, Inner Joins, Outer Joins, Antijoins, Semijoins*), powoduje, że pozornie proste zagadnienia stają się często źródłem wielu błędów. Ważnym przykładem zastosowania połączeń są też tzw. zapytania hierarchiczne, w których połączenia występują w sposób niejawni. Sprawę komplikuje ponadto wprowadzona niedawno (począwszy od wersji 9i) obsługa nowej składni połączeń wynikającej z implementacji w bazie Oracle kolejnych elementów standardu SQL/99. Niektóre opisane tam możliwości nie były ponadto (w sposób bezpośredni) dostępne we wcześniejszych wersjach bazy. W referacie przedstawiono opisane wyżej zagadnienia na odpowiednio dobranych przykładach. Pokazano również, że nawet pozornie proste zapytania kierowane do bazy mogą wymagać budowy ciekawych zapytań z połączeniami. Z uwagi na przyjęty zakres pracy, nie zajmowano się problemami „systemowej” optymalizacji zapytań. Pominięto więc takie zagadnienia jak np. indeksacja, partycjonowanie, widoki zmaterializowane, podpowiedzi dla optymalizator (ang. *hints*), itd.

Informacje o autorach:

dr inż. Artur Gramacki – pracuje w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego na stanowisku adiunkta. Jego zainteresowania koncentrują się wokół szeroko rozumianych zagadnień związanych z bazami danych, w szczególności firmy Oracle. Oprócz prowadzenia zajęć dydaktycznych stara się wykorzystywać swoją wiedzę uczestnicząc w różnych projektach informatycznych z tego zakresu. Brał udział w trzech projektach, których celem było przygotowanie systemów wspomagających działalność Uniwersytetu Zielonogórskiego.

dr inż. Jarosław Gramacki – jest pracownikiem naukowym w Instytucie Informatyki i Elektroniki Uniwersytetu Zielonogórskiego. Zajmuje się projektowaniem, wykonywaniem oraz wdrażaniem aplikacji bazodanowych usprawniających szeroko rozumiane zarządzanie Uczelnią. Od wielu lat prowadzi również zajęcia dydaktyczne dotyczące projektowania baz danych, działania i administrowania systemami zarządzania bazami danych oraz wykorzystania technologii Oracle w budowie aplikacji użytkowych.

1. Wstęp

Dane zgromadzone w bazie danych są użyteczne tylko wówczas, gdy potrafimy z nich poprawnie i bezbłędnie skorzystać. Jedyną metodą dostępu do danych bazy relacyjnej jest język SQL i to niezależnie od platformy programistycznej, z której korzystamy [ORASQL]. Zapytania w tym języku pojawiają się zawsze – nawet, gdy są bardzo skrytycznie ukryte przed użytkownikiem.

Ważnym zagadnieniem w ramach języka SQL są połączenia relacji (tabel) pomiędzy sobą i w konsekwencji obsługa tychże połączeń z poziomu języka SQL. Niniejszy artykuł poświęcony jest w całości omówieniu wybranych aspektów połączeń w relacyjnej bazie danych Oracle.

2. Model danych¹

Przy opisie jakichkolwiek zagadnień dotyczących języka SQL konieczny jest odpowiedni, adekwatny do opisywanych problemów, schemat tabel relacyjnych wraz z ich powiązaniem. Na rysunku 1 przedstawiono wykorzystywany w pracy model relacyjny. Większość tabel tworzy klasyczny wręcz model używany w wielu pracach dotyczących relacyjnych baz danych i pochodzi z oryginalnego schematu demonstracyjnego systemu Oracle o nazwie *HR* (ang. *human resources*) – szczegółowo omawiany jest on w [HRSchema]. Wymieńmy jednak kilka jego „niedogodności”, które będą istotne w pracy:

1. Kolumna `Employees.manager_ID` dopuszcza wartość `NULL`.
2. Kolumna `Employees.department_ID` dopuszcza wartość `NULL`.
3. Kolumna `Departments.manager_ID` dopuszcza wartość `NULL`.
4. Wygodna na etapie modelowania, ale sprawiająca trudności w użyciu i podatna na błędy, relacja wewnętrzna `Employees.employee_ID` -> `Employees.manager_ID`.
5. W tabeli `Sport_enrollment` (patrz następny akapit) pracownik może zapisać się do sekcji sportowej, która akurat nie jest prowadzona w danym sezonie (np. sekcja tenisa jest oferowana w sezonie letnim).

Na potrzeby pracy rozszerzono ten model². Tabele `Sports` oraz `Sport_enrollment` (nie występują one w oryginalnym schemacie *HR*) opisują prowadzone w firmie „ligi” sportowe i pracowników, którzy zapisali się na dane rozgrywki. Potrójny klucz *primary* (na wszystkich kolumnach tabeli `Sport_enrollment`) utworzono aby „bezkosztowo” uniemożliwić wielokrotne zapisanie się tego samego pracownika na te same rozgrywki.

Część z wymienionych wyżej właściwości (1, 2, 3) może być efektem celowo przyjętych założeń i z formalnego punktu widzenia nie stanowi błędu.

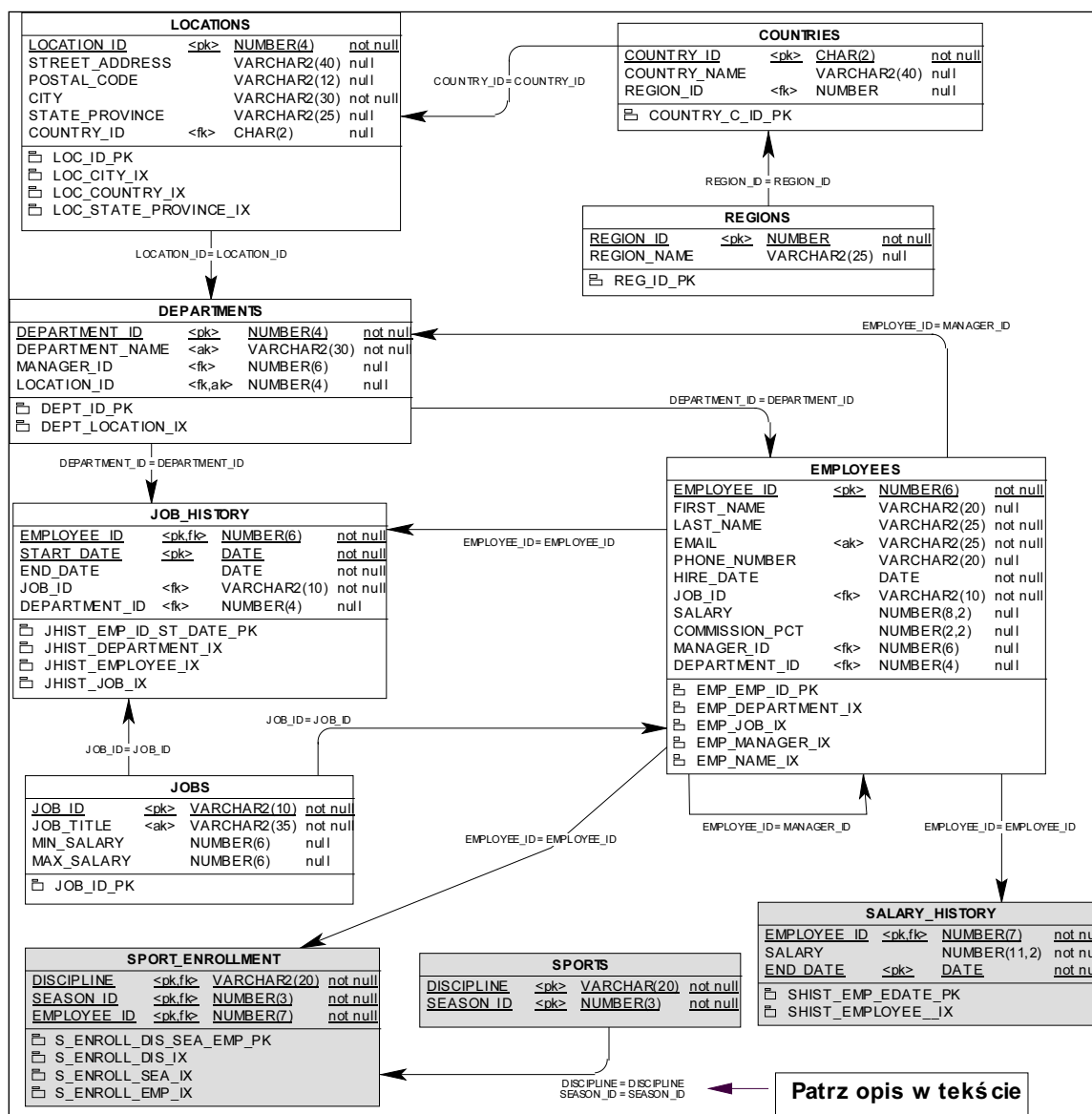
Właściwość 4 można by zastąpić odpowiednią strukturą typu *master-detail* (mniej podatna na błędy), jednak wtedy należałoby z góry ustalić ilość poziomów podległości pracowników.

Właściwość 5 jest wynikiem pewnego niedbalstwa projektanta bazy. Bez problemu można było utworzyć odpowiednie ograniczenie, np. poleceniem:

```
ALTER TABLE Sport_enrollment
ADD CONSTRAINT xxx_FK FOREIGN KEY (discipline, season_ID)
REFERENCES Sports (discipline, season_ID);
```

¹ W pracy wszystkie nazwy tabel pisane są małymi literami + pierwsza litera duża (np. `Departments`), nazwy kolumn w całości małymi literami (np. `title`), aliasy dla kolumn to jedna bądź dwie duże litery (np. `J`, `SE`).

² Wszystkie skrypty wykorzystywane w pracy można otrzymać kontaktując się z autorami.



Rys. 1. Model relacyjny HR uzupełniony dodatkowe tabele (zaznaczone ciemniejszym kolorem)

Ponieważ jednak do czasu zauważenia błędu wprowadzono (tak zakładamy) dużą ilość „brudnych” danych (pracownicy pozapisywali się do sekcji, do których chcieliby należeć w wygodnych dla nich okresach, interpretując wprowadzone wpisy jako swoistą listę życzeń) więc dodanie ograniczenia nie wchodzi (przynajmniej na razie) w grę. Innym, często spotykanym w praktyce powodem niemożności usunięcia opisanego błędu jest brak odpowiednich uprawnień. Osobami, które najczęściej budują zapytania do bazy są programiści raportów, którzy w poprawnie administrowanym środowisku mają jedynie uprawnienia `SELECT` do obiektów schematu.

W stosunku do oryginalnego schematu *HR* usunięto również występujące w nim jawne „błędy” projektowe. W tabeli `Departments` dodano ograniczenie `unique` dla kolumn `department_name` oraz `location_id`. W tabeli `Jobs` dodano ograniczenie `unique` dla kolumny `job_title`. Oba ograniczenia oznaczono na schemacie symbolem `<ak>`.

Dodano również tabelę `Salary_history`. W oryginalnym schemacie *HR* można zapamiętywać historię zmian miejsca pracy pracownika (kolumna `Employees.department_id`) lub zmian stanowiska (kolumna `Employees.job_id`). Jest to implementacja tzw. wersjowania relacji. Brakuje natomiast możliwości zapamiętywania np. zmian płacy pracownika (kolumna `Employ-`

ees.salary), czyli tzw. wersjowania atrybutów. Obsługę tą zapewnia właśnie tabela `Salary_history`. Dane do niej zapisywane są z użyciem analogicznego wyzwalacza do tego, który w oryginalnej wersji schematu HR obsługuje wpisy do tabeli `Job_history`.

Abstrahując jednak od powyższej dyskusji, w praktyce często przychodzi nam pracować na modelu takim jaki jest (nie mamy możliwości zmiany modelu). Dużego znaczenia nabiera wtedy poprawne konstruowanie zapytań opartych na danym modelu.

3. Połączenia w systemie Oracle

Połączenia zajmują ważne miejsce w każdej bazie relacyjnej i są to bardzo newralgiczne elementy. Dokumentacja systemu Oracle poświęca im jednak stosunkowo niewiele miejsca, dodatkowo prawie zupełnie nie zwracając uwagi na różne związane z nimi niuanse, gdyż prezentowane tam przykłady są typu „zawsze zadziała”. Ponadto w oryginalnej dokumentacji systemu Oracle pojawia się siedem pojęć, w których używana jest fraza „join”³, zestawiono je poniżej. Zdaniem autorów część z nich, jakkolwiek poprawna, wprowadza zbędne zamieszanie w i tak już skomplikowanej materii. W nawiasach podano również inne, pojawiające się w różnych publikacjach nazwy, które jednak nie pojawiają się w dokumentacji Oracle. W Tabeli 1 krótko opisano każde z połączeń:

- *(Natural Joins)*,
- *Equijoins*,
- *Self Joins*,
- *Cartesian Products*,
- *Inner Joins (Simple Joins)*,
- *Outer Joins*,
- *Antijoins*,
- *Semijoins*,
- *(Theta Joins)*.

Tabela 1. Różne używane pojęcia związane z połączeniami relacyjnymi

Nazwa	Opis
<i>Natural Joins</i>	Połączenie dwóch lub więcej tabel poprzez kolumny o tych samych nazwach. W praktyce chodzi z reguły o połączenia równościowe.
<i>Equijoins</i>	Dotyczy połączenia opisanego operatorem porównania. Nie wspomina się o żadnych dodatkowych zagadnieniach / założeniach szczegółowych.
<i>Self Joins</i>	Dotyczy połączenia w obrębie jednej tabeli. Tabela ta musi więc dwa razy pojawić się w klauzuli <code>FROM</code> (z reguły z dwoma różnymi aliasami). Nie wspomina się tu jednak o ew. strukturze hierarchicznej, która z reguły jest opisywana poprzez relację zdefiniowaną w jednej tabeli dla dwóch kolumn (a te wymagają z reguły zastosowania specjalizowanych operatorów).

³ Wszędzie użyto pisowni zgodnej z używaną w oryginalnej dokumentacji firmy Oracle.

Nazwa	Opis
<i>Cartesian Products</i>	Wynik połączenia dwóch lub więcej tabel, dla których nie podano warunku połączenia. Każdy wiersz z jednej tabeli połączony zostanie z każdym wierszem drugiej tabeli. Poza rzadkimi przypadkami szczególnymi, jest to z reguły wynik błędu w zapytaniu.
<i>Inner Joins</i> (<i>Simple Joins</i>)	Dosyć ogólne pojęcie. Połączenie dwóch lub więcej tabel, które zwraca w wyniku tylko wiersze pasujące do warunku połączenia. Nie mówi się nic o obsłudze wierszy „niepasujących”, które z reguły należy złączyć zewnętrznie (<i>Outer Joins</i>)
<i>Outer Joins</i>	Rozszerzenie wyniku połączenia typu <i>Inner Joins</i> o wiersze „niepasujące” Rozróżnia się połączenia zewnętrzne lewostronne (<i>Left Outer Joins</i>), prawostronne (<i>Right Outer Joins</i>), pełne (<i>Full Outer Joins</i>).
<i>Antijoins</i>	Połączenie zwracające wiersze, które NIE pasują do podanego warunku.
<i>Semijoins</i>	Połączenie z warunkiem EXISTS. Duplikaty zostają usunięte.
<i>Theta Joins</i>	Połączenie nierównościowe. Połączenie dwóch relacji dla kryterium połączenia innym niż równość.

Z analizy zawartości Tabeli 1 wynika, że istnienie przynajmniej kilku z podanych tam pojęć jest dyskusyjne, gdyż służą one do opisu stosunkowo wąskiego zbioru zapytań. Z drugiej strony niektóre nazwy opisują bardzo ogólne właściwości, typowe raczej dla całego podejścia relacyjnego. Zebrano je jednak w pracy dla pokazania (zdaniem autorów nieuzasadnionego) trendu w tworzeniu nie zawsze ułatwiających życie, „bytów”.

4. Przykłady

Z uwagi na ograniczoną objętość pracy, w pozostałej jej części zamieszczono jedynie pewną liczbę przykładów ilustrujących wybrane zagadnienia dotyczące połączeń relacyjnych oraz krótkie komentarze tychże przykładów. Część przykładów zaczerpnięto z prac [Genni00] [Czupr03], dostosowując je do używanego w artykule schematu. Przykłady starano się dobrać tak, aby pojawiały się w nich „problemy”. Nie oznacza to oczywiście, że wyczerpano temat. Wręcz przeciwnie – jedynie zasygnalizowano go.

W poniższych przykładach, gdzie to tylko było możliwe, zastosowano nową (dostępną począwszy od wersji 9i) składnię połączeń zdefiniowaną w standardzie SQL/99⁴. Należy jednak zaznaczyć, że nowa składnia NIE wprowadza żadnej nowej funkcjonalności, której nie było w systemie do tej pory. Czyni jednak zapytania bardziej zgodnymi z normą ANSI-SQL⁵. Zapytanie takie uruchomione na innej platformie bazodanowej (zgodnej z ANSI-SQL) powinno (przynajmniej teoretycznie) zadziałać bez żadnych modyfikacji.

Przykład 1 – połączenia naturalne

Składnia dotychczasowa	Składnia ANSI
SELECT R.region_id, Rr.region_name, C.country_id,	SELECT region_id, R.region_name, C.country_id,

⁴ W wielu publikacjach pojawia się taka właśnie nazwa. Należy ją jednak traktować jako umowny skrót. Równie dobry skrót byłby SQL/2003 Poprawna nazwa normy to: ANSI/ISO/IEC 9075-[tu numer od 1 do 14]:2003

⁵ Norma dostępna na stronie <http://webstore.ansi.org/ansidocstore/default.asp> za \$125 !

<pre> C.country_name FROM Countries C, Regions R WHERE C.region_id = R.region_id AND LOWER(R.region_name) LIKE '%europe%'; </pre>	<pre> C.country_name FROM Countries C NATURAL JOIN Regions R WHERE LOWER(R.region_name) LIKE '%europe%'; </pre>
---	--

Komentarz:

- Dla składni ANSI, często w takim przypadku pojawia się błąd: *ORA-25155: column used in NATURAL join cannot have qualifier*. Stąd zamiast `SELECT R.region_id` musi być `SELECT region_id` (nawet pomimo istnienia aliasów tabel).
- Wydaje się, że używanie klauzuli `NATURAL JOIN`, NIE powinno być zalecane. Łatwo bowiem o (trudny do wykrycia) błąd, który wystąpi po dodaniu do schematu kolumny, która przypadkowo będzie nazywała się tak samo, jak kolumna w zapytaniu używającym `NATURAL JOIN`. System Oracle implementuje tę składnię, ale wynika to chyba raczej z potrzeby zapewnienia zgodności ze standardem ANSI, którego użyteczność w przypadku `NATURAL JOIN` jest jednak mocno dyskusyjna.
- Nowa składnia umożliwi rozdzielenie warunków połączenia od innych warunków w klauzuli `WHERE` – tu ograniczających wyniki do regionów europejskich. W dotychczasowej składni oba występowały w klauzuli `WHERE` i dla złożonych zapytań trudno było odróżnić je od siebie.

Przykład 2 – nowa składnia połączeń, pierwsza wersja

Składnia dotychczasowa	Składnia ANSI
<pre> SELECT D.department_name, E.last_name ' ' e.first_name Name, E.hire_date FROM Employees E, Departments D WHERE E.department_id = D.department_id AND E.job_id = 'IT_PROG'; </pre>	<pre> SELECT D.department_name, E.last_name ' ' e.first_name Name, E.hire_date FROM Employees E JOIN Departments D USING(department_id) WHERE E.job_id = 'IT_PROG'; </pre>

Komentarz:

- Jeśli w zapytaniu ANSI zamiast:

```
FROM Employees E JOIN Departments D USING(department_id)
```

zapiszemy:

```
Employees E NATURAL JOIN Departments D
```

otrzymamy zły wynik. W powyższym przykładzie natura zagrożenia jest inna niż zagrożenie sygnalizowane w Przykładzie 1. Tabele `Employees` oraz `Departments` łączą DWIE relacje i dlatego należy doprecyzować, o które połączenie chodzi. Umożliwia to klauzula `USING`.
- Klauzuli `USING` można używać tylko wówczas, gdy odpowiednie kolumny obu tabel mają takie same nazwy.
- Tradycyjna składnia jest wolna od takich zagrożeń, gdyż warunek połączenia podany jest jawnie.
- Klauzula `USING` subtelnie „psuje” jednak semantykę zapytania. Nie uda się np. `SELECT D.department_name, D.department_ID, E.department_ID, ...` gdyż zapytanie zakończy się błędem: *ORA-25154 -column part of USING clause cannot have qualifier*. W tradycyjnej składni błąd ten nie wystąpi.

- W klauzuli `USING` nie ma ograniczenia na ilość kolumn w niej wymienianych. Zapytanie jak poniżej jest jak najbardziej poprawne:

```
SELECT
  discipline, -- Błąd, gdy S.discipline - patrz opis wyżej
  season_ID,
  SE.employee_ID
FROM
  Sports S INNER JOIN Sport_enrollment SE
  USING (discipline, season_id);
```

- Zamiast `JOIN` można użyć dłuższej, ale bardziej opisowej wersji `INNER JOIN`, która podkreśla, że chodzi o złączenie wewnętrzne (w odróżnieniu od omawianych za chwilę złączeń zewnętrznych). Oracle nie wymusza jednak tego, co jest niekonsekwencją, gdyż w złączeniach zewnętrznych odpowiednie słowo (`OUTER JOIN`), jest obowiązkowe. Należy więc przypuszczać, że wśród programistów zapanuje duża dowolność w jego stosowaniu.
- Klauzula `USING` istotnie zwiększa czytelność zapytania.

Przykład 3a – nowa składnia połączeń, druga wersja

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT D.department_name, E.last_name ' ' e.first_name Name, S.discipline, S.season_ID FROM Employees E, Departments D, Sport_enrollment S WHERE E.department_id = D.department_id AND E.employee_ID = S.employee_ID AND E.job_id = 'IT_PROG';</pre>	<pre>SELECT D.department_name, E.last_name ' ' e.first_name Name, S.discipline, S.season_ID FROM Employees E INNER JOIN Departments D ON (E.department_id = D.department_id) INNER JOIN Sport_enrollment S ON (E.employee_ID = S.employee_ID) WHERE E.job_id = 'IT_PROG';</pre>

Komentarz:

- Sens zastosowania klauzuli `ON` sprowadza się jedynie do (eleganckiego) rozdzielenia warunków połączenia od innych warunków logicznych. W tradycyjnej składni wszystkie warunki występowały razem w ramach klauzuli `WHERE`.

Przykład 3b – nowa składnia połączeń, porównanie `ON` oraz `USING`

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S, Sport_enrollment SE WHERE S.discipline = SE.discipline AND S.season_id = SE.season_id;</pre>	<pre>-- wersja z USING SELECT discipline, -- tu alias zabroniony season_ID, -- tu alias zabroniony SE.employee_ID FROM Sports S INNER JOIN Sport_enrollment SE USING (discipline, season_id) -- wersja z ON SELECT S.discipline, -- tu alias obowiązkowy S.season_ID, -- tu alias obowiązkowy SE.employee_ID FROM Sports S INNER JOIN Sport_enrollment SE ON S.discipline = SE.discipline AND S.season_id = SE.season_id;</pre>

Komentarz:

- Wersje z USING i ON są w tym przypadku równoważne, gdyż odpowiednie kolumny łączonych tabel mają takie same nazwy. W praktyce nie zawsze jednak ma to miejsce.
- Klauzula ON jest mniej czytelna od klauzuli USING. Mając wybór lepiej zastosować USING.
- W klauzuli ON może pojawić się dowolne wyrażenie logiczne. Niemniej w praktyce większość zapytań to zapytania równościowe (*Equijoins*), w których porównujemy odpowiednie kolumny dwóch tabel.
- zwracamy uwagę na obowiązkowość / niedozwoloność stosowania aliasów przy nazwach kolumn.

Przykład 3c – nowa składnia połączeń, połączenia wielokrotne

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT S.discipline, S.season_ID, E.employee_ID, -- tu alias konieczny E.last_name FROM Sports S, Sport_enrollment SE, Employees E WHERE S.discipline = SE.discipline AND S.season_id = SE.season_id AND E.employee_ID = SE.employee_ID</pre>	<pre>SELECT discipline, season_ID, employee_ID, -- teraz tu alias zabroniony E.last_name FROM Sports S INNER JOIN Sport_enrollment SE USING (discipline, season_id) INNER JOIN Employees E USING (employee_ID)</pre>

Komentarz:

- Połączenia wielokrotne są możliwe. INNER JOIN uwypukla kolejność połączeń wielokrotnych.
- Oracle domyślnie wykonuje połączenia w kierunku od lewej do prawej. Najpierw połączenie Sports z Sport_enrollment a następnie wynik tego połączenia z Employees. Można to jednak zmienić stosując w odpowiednim miejscu nawiasy.

Przykład 4 - połączenia nierównościowe

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT E.job_id, J.job_title, E.first_name '-' E.last_name Name, J.min_salary '--' J.max_salary "Przedział", E.salary FROM Employees E, Jobs J WHERE E.salary BETWEEN J.min_salary AND J.max_salary ORDER BY E.last_name;</pre>	<pre>SELECT E.job_id, J.job_title, E.first_name '-' E.last_name Name, J.min_salary '--' J.max_salary "Przedział", E.salary FROM Employees E INNER JOIN Jobs J ON (E.salary BETWEEN J.min_salary AND J.max_salary) ORDER BY E.last_name;</pre>

Komentarz:

- Przykład pokazuje połączenie nierównościowe (ang. *Theta Joins*). Zdecydowanie rzadziej używane niż połączenia równościowe.
- W powyższym przykładzie wyświetlono listę pracowników oraz, na bazie tabeli Jobs, sprawdzono, czy zarobki poszczególnych pracowników mieszczą się w „widełkach”. Można przykładowo zauważyć, że pracownik o nazwisku *Abel*, pracujący na stanowisku *Sales Re-*

presentative (tego jednak z przestawionego zapytania nie odczytamy) zarabia kwotę 11000, która to kwota mieści się w przedziale dla pięciu innych stanowisk.

- Połączenie jest bardzo „pracochłonne” do wykonania. Wymaga najpierw utworzenia iloczynu kartezyjskiego a następnie wybraniu z niego wierszy, dla których zadany warunek jest spełniony.

```
-- Wynik zapytania
```

JOB_ID	JOB_TITLE	NAME	Przedział	SALARY
SA_REP	Finance Manager	Ellen Abel	8200 -- 16000	11000,00
SA_REP	Accounting Manager	Ellen Abel	8200 -- 16000	11000,00
SA_REP	Sales Representative	Ellen Abel	6000 -- 12000	11000,00
SA_REP	Sales Manager	Ellen Abel	10000 -- 20000	11000,00
SA_REP	Marketing Manager	Ellen Abel	9000 -- 15000	11000,00
SA_REP	Purchasing Manager	Ellen Abel	8000 -- 15000	11000,00
SA_REP	Accountant	Sundar Ande	4200 -- 9000	6400,00
SA_REP	Public Accountant	Sundar Ande	4200 -- 9000	6400,00
...				

646 rows selected

(Dla iloczynu kartezyjskiego - 2033 rekordów)

Przykład 5 - połączenia *Antijoins*

Składnia dotychczasowa	Składnia ANSI
<pre>-- Antijoins SELECT E.job_id, E.first_name ' ' E.last_name Name, E.salary FROM Employees E WHERE department_id NOT IN (SELECT department_id FROM Departments WHERE location_id BETWEEN 1700 AND 2700); -- Equijoins SELECT E.job_id, E.first_name ' ' E.last_name Name, E.salary FROM Employees E, Departments D WHERE E.department_id = D.department_id AND D.location_id NOT BETWEEN 1700 AND 2700;</pre>	<pre>-- Antijoins Nie dotyczy -- Equijoins SELECT E.job_id, E.first_name ' ' E.last_name Name, E.salary FROM Employees E INNER JOIN Departments D USING (department_ID) WHERE D.location_ID NOT BETWEEN 1700 AND 2700;</pre>

Plany wykonania
<pre>-- Antijoins SELECT STATEMENT FILTER TABLE ACCESS FULL EMPLOYEES TABLE ACCESS BY INDEX ROWID DEPARTMENTS INDEX RANGE SCAN DEPT_LOCATION_IX -- Equijoins SELECT STATEMENT CONCATENATION TABLE ACCESS BY INDEX ROWID EMPLOYEES</pre>

NESTED LOOPS	TABLE ACCESS INDEX	BY INDEX ROWID RANGE SCAN	DEPARTMENTS DEPT_LOCATION_IX
INDEX RANGE SCAN	TABLE ACCESS NESTED LOOPS	BY INDEX ROWID	EMP_DEPARTMENT_IX EMPLOYEES
TABLE ACCESS INDEX	BY INDEX ROWID RANGE SCAN	DEPARTMENTS DEPT_LOCATION_IX	
INDEX RANGE SCAN		EMP_DEPARTMENT_IX	

Komentarz:

- Przykład pokazuje zapytanie, w którym interesuje nas BRAK zachodzenia danej relacji (połączenie *Antijoins*). Pokazano pracowników pracujących na wydziałach zlokalizowanych w wybranych lokalizacjach (poprzez negacje pozostałych).
- Dla *Antijoins* występuje podzapytanie. Nowa składnia jest więc bezprzedmiotowa.
- Ten sam wynik osiągnąć można stosując zwykłe połączenia równościowe dwóch tabel.
- Plan wykonania zapytania *Equijoins* w stosunku do planu zapytania *Antijoins* będzie jednak bardziej złożony. W planie dla *Antijoins* występuje jednakże operacja TABLE ACCESS FULL. W konkurencyjnym planie wszędzie korzystamy z indeksów.
- Decyzję, czy użyć połączenia *Antijoins*, czy zwykłego równościowego *Equijoins* powinna poprzedzić analiza planu zapytania pod kątem ewentualnych korzyści / strat.

Do tej pory nie zwracano uwagi na fakt, że wiele wierszy będących *de facto* oczekiwanym wynikiem zapytania, NIE zostanie wyświetlonych. Z reguły będzie tak z powodu istnienia pewnej ilości kolumn <foreign key> dopuszczających wartości NULL. W efekcie pewna ilość wierszy nie będzie spełniała warunków połączenia i nie pojawi się na ekranie. Błędy wynikające z istnienia w schemacie wspomnianych kolumn NULL są bardzo trudne do wykrycia, szczególnie gdy zapytanie zwraca dużą ilość danych.

Do obsługi takich przypadków od dość dawna stosuje się tzw. złączenia zewnętrzne w kilku wersjach: lewostronne, prawostronne, pełne (ang. *left outer join*, *right outer join*, *full outer join*). Należy podkreślić, że system Oracle wraz z pojawieniem się wersji 9i nie wprowadza jeśli chodzi o obsługę takich połączeń żadnych merytorycznych zmian⁶. Istotnie jednak zmienia (upraszcza) odpowiednią do obsługi takich przypadków składnię SQL.

Przykład 6 – prawostronne połączenia zewnętrzne

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT NVL(D.department_name, '???') Dept, SUM(E.salary) Sal FROM Employees E, Departments D WHERE D.department_id(+) = E.department_id GROUP BY D.department_name;</pre>	<pre>SELECT NVL(D.department_name, '???') Dept, SUM(E.salary) Sal FROM Departments D RIGHT OUTER JOIN Employees E ON D.department_id = E.department_id GROUP BY D.department_name;</pre>

Komentarz:

- Intencją zapytania było podsumowanie kosztów płacowych wydziałów. Brak prawostronnego złączenia zewnętrznego NIE uwzględni wypłat pracowników, którzy NIE są przypisani do jakiegokolwiek wydziału.

⁶ W dokumentacji wymienione są jednak pewne ograniczenia składni z (+), których nie ma w składni ANSI. Szczegóły patrz w [ORASQL].

- W przypadku tradycyjnej składni kolejność kolumn w klauzuli WHERE jest nieistotna. Wersja WHERE E.department_id = D.department_id(+) też będzie poprawna. Niestety wersja z warunkiem:
FROM Employees E RIGHT OUTER JOIN Departments D
da już zupełnie inny wynik. Wynik będzie taki jak dla lewostronnego złączenia zewnętrznego czyli tak, jakby napisano:
FROM Employees E LEFT OUTER JOIN Departments D.
- Zamiana na ON E.department_id = D.department_id nie wprowadza podobnego niebezpieczeństwa.
- Podsumowania (klauzula GROUP BY) mogą w niezamierzony sposób zamaskować zarówno błędy w złączeniach relacyjnych jak i inne „potknięcia” programisty w klauzuli WHERE. W tym konkretnym przypadku błąd wynikający z braku (+) łatwo przeoczyć, gdyż liczba danych w tabelach jest stosunkowo duża i błędy nie są na pierwszy rzut oka widoczne. Podsumowywać więc należy tylko wtedy, gdy jesteśmy pewni, że podsumowujemy właściwe dane.

Przykład 7 – lewostronne połączenia zewnętrzne

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT D.department_name Dept, NVL(SUM(E.salary), 0) Sal FROM Employees E, Departments D WHERE D.department_id = E.department_id(+) GROUP BY D.department_name;</pre>	<pre>SELECT D.department_name Dept, NVL(SUM(E.salary), 0) Sal FROM Departments D LEFT OUTER JOIN Employees E ON D.department_id = E.department_id GROUP BY D.department_name;</pre>

Komentarz:

- Intencją zapytania było wyświetlenie kosztów płacowych WSZYSTKICH wydziałów. Brak lewostronnego złączenia zewnętrznego NIE uwzględni na liście wydziałów, w których nikt nie pracuje.
- W przeciwieństwie do poprzedniego przykładu, problem jest tutaj innej natury. Dane, które otrzymamy, są co prawda poprawne (suma zgadza się), jednak błędny jest ich charakter informacyjny („puste” wydziały nie będą widoczne).
- Uwagi dotyczące kolejności argumentów operatora są takie same, jak w poprzednim przykładzie.

Przykład 7b – lewostronne połączenia zewnętrzne

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S, Sport_enrollment SE WHERE S.discipline = SE.discipline(+) AND S.season_id = SE.season_id(+);</pre>	<pre>SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S LEFT OUTER JOIN Sport_enrollment SE ON S.discipline = SE.discipline AND S.season_id = SE.season_id;</pre>

Komentarz:

- W wyniku wykonania zapytania pojawi się lista sekcji sportowych zarówno obsadzonych przez chociaż jednego pracownika, jak i tych, do których nie zapisał się żaden pracownik.

- Tabela `Sport_enrollment` powinna zawierać podwójny klucz obcy:

```
ALTER TABLE Sport_enrollment
ADD CONSTRAINT xxx_FK FOREIGN KEY (discipline, season_ID)
REFERENCES Sports (discipline, season_ID);
```

Jak zostało jednak wspomniane na początku artykułu, nie został on w rzeczywistości utworzony lub jest nieaktywny. Fakt ten zostanie wykorzystany w kolejnym przykładzie. Obecny przykład pokazuje jedynie, że nie ma ograniczenia na ilość zewnętrznie łączonych kolumn z użyciem `ON`.

Przykład 8 – pełne połączenia zewnętrzne

Składnia dotychczasowa	Składnia ANSI
<pre>SELECT D.department_name Dept, NVL(SUM(E.salary), 0) FROM Employees E, Departments D WHERE D.department_id = E.department_id(+) GROUP BY D.department_name UNION SELECT NVL(D.department_name, '???') Dept, SUM(E.salary) FROM Employees E, Departments D WHERE D.department_id(+) = E.department_id GROUP BY D.department_name;</pre>	<pre>SELECT NVL(D.department_name, '???') Dept, NVL(SUM(E.salary), 0) Sal FROM Departments D FULL OUTER JOIN Employees E ON D.department_id = E.department_id GROUP BY D.department_name;</pre>

Komentarz:

- Powyższy przykład jest jakby połączeniem poprzednich zapytań ilustrujących lewo i prawostronne połączenia zewnętrzne.
- W przeciwieństwie do poprzednich przykładów, nowa składnia ANSI istotnie upraszcza zapis.
- W przypadku tradycyjnej składni mamy możliwość użycia `UNION` lub `UNION ALL`. Nowa składnia NIE daje możliwości takiego wyboru. W wersji z `UNION ALL` otrzymamy co prawda wiele duplikatów, jednak zapytanie zostanie wykonane dużo bardziej efektywnie niż wersja z `UNION`, gdyż nie wykonuje się kosztowne `SORT UNIQUE`. Duplikaty można następnie próbować usunąć „lokalnie”, np. w aplikacji raportującej.
- Uwaga! Zgodnie z teorią relacyjną zamiana `UNION` i `UNION ALL` jest możliwa tylko wtedy, gdy `UNION`-owane zbiory nie mają wspólnych wierszy. W naszym przypadku warunek ten NIE występuje, jednak zgodnie z poprzednią uwagą „zgodziliśmy” się na duplikaty. Warunek taki spełnia na przykład poniższe zapytanie. Tu bezwzględnie preferowane będzie użycie `UNION ALL`:

```
SELECT
  department_name, department_ID FROM Departments
WHERE
  department_ID > 100

UNION ALL
```

```

SELECT
  department_name, department_ID FROM Departments
WHERE
  department_ID <= 100
ORDER BY 1;

```

Przykład 8b – pełne połączenia zewnętrzne, wersje z USING oraz ON

Składnia dotychczasowa	Składnia ANSI
<pre> SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S, Sport_enrollment SE WHERE S.discipline = SE.discipline(+) AND S.season_id = SE.season_id(+) UNION SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S, Sport_enrollment SE WHERE S.discipline(+) = SE.discipline AND S.season_id(+) = SE.season_id; </pre>	<pre> -- z USING -- wynik "BŁĘDNY" SELECT discipline, season_ID, employee_ID FROM Sports S FULL OUTER JOIN Sport_enrollment SE USING (discipline, season_id) -- z ON -- wynik "POPRAWNY" SELECT S.discipline, S.season_ID, SE.employee_ID FROM Sports S FULL OUTER JOIN Sport_enrollment SE ON S.discipline = SE.discipline AND S.season_id = SE.season_id; </pre>

Komentarz:

- Zawartość tabel Sports oraz Sport_enrollment jest następująca:

```

select * from sports;

DISCIPLINE          SEASON_ID
-----
football            1
football            4
hockey              1
hockey              3
tennis              1
tennis              2
tennis              5
volleyball          1
volleyball          2
volleyball          4

10 rows selected

select * from sport_enrollment;

DISCIPLINE          SEASON_ID EMPLOYEE_ID
-----
hockey              1          100
tennis              2          200
tennis              10         105

```

- Jak widać pracownik o numerze 105 zapisał się „z wyprzedzeniem” do sekcji tenisa na sezon 10. W tym sezonie tenis nie jest jednak (jeszcze) oferowany. Było to możliwe, gdyż nie istnieje odpowiedni klucz obcy lub jest on nieaktywny (była o tym mowa wcześniej).
- Wiele ofert uprawiania sportu nie zostało jeszcze wybranych. Aby pokazać je na wydruku zastosowano zwykłe pełne złączenie zewnętrzne FULL OUTER JOIN.

- W wersji zapytania z USING konsekwencją użycia klauzuli USING jest w pewnym sensie „błędny” wynik. Jest on oczywiście zgodny z wpisanymi do tabel danymi, jednak niepoprawny merytorycznie, gdyż na jego podstawie otrzymamy mylące informacje. Dla wiersza „tennis 10 105” nie ma odpowiedniego wiersza w tabeli Sports. Warto również zauważyć, że nie mamy tu do czynienia z wartościami NULL w kolumnach. W klauzuli USING podano jedynie podług których kolumn ma odbywać się połączenie.

```

SELECT discipline, season_ID, employee_ID
FROM Sports S FULL OUTER JOIN Sport_enrollment SE
USING (discipline, season_id);

DISCIPLINE          SEASON_ID EMPLOYEE_ID
-----
hockey                1          100
tennis                2          200
volleyball            4
football              1
tennis                1
football              4
volleyball            1
hockey                3
tennis                5
volleyball            2
tennis                10         105

11 rows selected

```

- W wersji zapytania z ON otrzymujemy wynik „poprawny”. Tu z kolei nie jest on zgodny z wpisanymi w bazie danymi, jednak jest poprawny merytorycznie (nie można zapisać się do nie oferowanej w danym sezonie sekcji).

```

SELECT S.discipline, S.season_ID, SE.employee_ID
FROM Sports S FULL OUTER JOIN Sport_enrollment SE
ON S.discipline = SE.discipline AND
S.season_id = SE.season_id;

DISCIPLINE          SEASON_ID EMPLOYEE_ID
-----
hockey                1          100
tennis                2          200
tennis                5
volleyball            4
volleyball            2
football              1
football              4
volleyball            1
hockey                3
tennis                1
                                105

11 rows selected

```

- W tradycyjnej składni opisane wyżej podobne „dylematy” nie mają miejsca. Zawsze otrzymujemy wynik jak dla wersji z ON.

Przykład 9 – iloczyn kartezjański

Składnia dotychczasowa	Składnia ANSI
<pre> SELECT D.department_name, L.city, FROM departments D, locations L; </pre>	<pre> SELECT D.department_name, L.city FROM departments D CROSS JOIN locations L; </pre>

Komentarz:

- Iloczyn kartezjański najczęściej pojawia się jako niezamierzony wynik (brak jakiegoś koniecznego połączenia). Nowa składnia wymaga jawnego wskazania, że intencją programisty był iloczyn kartezjański.
- Ponieważ jednak tradycyjna składnia ciągle jest (i pewnie będzie) obsługiwana przez bazę Oracle, więc praktyczna użyteczność `CROSS JOIN` w sensie eliminacji błędów jest dyskusyjna.

Przykład 10 – *Self Joins*

```
SELECT employee_ID, last_name, job_ID, manager_id FROM Employees;

-- Fragment zawartości tabeli Employees
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
100	King	AD_PRES	
101	Kochhar	AD_VP	100
102	De Haan	AD_VP	100
103	Hunold	IT_PROG	102
104	Ernst	IT_PROG	103
105	Austin	IT_PROG	103
106	Pataballa	IT_PROG	103
107	Lorentz	IT_PROG	103
114	Raphaely	PU_MAN	100
115	Khoo	PU_CLERK	114
122	Kaufling	ST_MAN	100
146	Partners	SA_MAN	100
147	Errazuriz	SA_MAN	100
148	Cambrault	SA_MAN	100
156	King	SA_REP	146
173	Kumar	SA_REP	148
...			

107 rows selected

```
-- Składnia tradycyjna
-- Dobrze

SELECT
  E1.employee_ID,
  E1.last_name||''''s boss: '||E2.last_name Kto_pod_kim,
  E1.manager_ID
FROM
  Employees E1, Employees E2
WHERE
  E2.employee_ID = E1.manager_ID AND E1.last_name LIKE 'K%'

EMPLOYEE_ID KTO_POD_KIM                                MANAGER_ID
-----
122 Kaufling's boss: King                               100
115 Khoo's boss: Raphaely                               114
156 King's boss: Partners                               146
101 Kochhar's boss: King                                100
173 Kumar's boss: Cambrault                             148

5 rows selected

-- Składnia ANSI
-- Zle

SELECT
  E1.employee_ID,
  E1.last_name||''''s boss: '||E2.last_name Kto_pod_kim,
  E2.employee_ID
FROM
  Employees E1, Employees E2
WHERE
```

```

E1.employee_ID = E2.manager_ID AND E1.last_name LIKE 'K%'
EMPLOYEE_ID KTO_POD_KIM EMPLOYEE_ID
-----
122 Kaufling's boss: Mallin 133
122 Kaufling's boss: Rogers 134
122 Kaufling's boss: Gee 135
122 Kaufling's boss: Philtanker 136
122 Kaufling's boss: Chung 188
122 Kaufling's boss: Dilly 189
122 Kaufling's boss: Gates 190
....
27 rows selected

```

Komentarz:

- Użyto dwóch aliasów tej samej tabeli – jest to obowiązkowe. Bardzo częsty błąd polega na późniejszym niewłaściwym użyciu tych aliasów. Porównaj wyniki obu zapytań.
- Możliwych błędów, będących efektem zamiany aliasów, jest więcej. Przykładowo kolejny, inny (jednak ciągle błędny) wynik otrzymamy, gdy ostatnia linia pierwszej wersji zapytania wyglądać będzie tak:
WHERE E2.employee_ID = E1.manager_ID AND **E2**.last_name LIKE 'K%'
- Użycie nowej składni jest w części dyskusyjne. Wersja z USING, nie działa niestety poprawnie. Jediną możliwością zwracającą poprawne wyniki jest użycie ON. Jak już jednak pokazano na wcześniejszych przykładach, użycie ON jedynie rozdziela warunki połączenia od innych warunków (np. zawężających) wyniki zapytania i w żadnym razie nie eliminuje wcześniej opisywanych błędów.

```

-- Zle
SELECT
  employee_ID,
  E1.last_name||''s boss: '||E2.last_name Kto_pod_kim,
  manager_ID
FROM
  Employees E1 INNER JOIN Employees E2 USING (employee_ID, manager_ID)
WHERE
  E1.last_name LIKE 'K%';

-- Dobrze
SELECT
  E1.employee_ID,
  E1.last_name||''s boss: '||E2.last_name Kto_pod_kim,
  E1.manager_ID
FROM
  Employees E1 INNER JOIN Employees E2 ON (E2.employee_ID = E1.manager_ID)
WHERE
  E1.last_name LIKE 'K%';

```

- Połączenia rekurencyjne w ramach jednej tabeli, jakkolwiek eleganckie, są bardzo podatne na błędy w zapytaniach.

Przykład 11 – zapytania hierarchiczne

```

SELECT
  E.employee_ID,
  E.manager_ID,
  RPAD( RPAD(' ', 2*(LEVEL)-1, '.')||E.first_name || ' '||E.last_name, 30),
  D.department_name
FROM
  Employees E, Departments D
WHERE
  E.department_ID = D.department_ID

```

```

START WITH
  E.employee_ID = 114
CONNECT BY
  E.manager_ID = PRIOR E.employee_ID;

0 rows selected

```

Komentarz:

- W wyniku wykonania zapytania nie zostanie zwrócony żaden wiersz. Powód - patrz komentarz przy danych zawartych w tabelce towarzyszącej przykładowi 10.
- Pierwszorzędne znaczenie ma tu kolejność wykonywanych przez Oracle czynności. Kolejność jest następująca:
 1. Połączenie jest materializowane co oznacza, że w pierwszej kolejności brane są pod uwagę warunki połączenia.
 2. Do zwróconych w punkcie 1. wierszy stosowana jest klauzula `CONNECT BY`.
 3. Do zwróconych w punkcie 2. wierszy stosowana jest klauzula `WHERE` z innymi niż warunki połączenia elementami.
- Zawsze należy więc sprawdzać, czy warunek połączenia nie „wyfiltrował” nam koniecznych w zapytaniu hierarchicznym danych (tzw. korzenia, od którego pobierane są dane w hierarchii zlokalizowane poniżej). W naszym przypadku „zniknął” pracownik o numerze 114 i w konsekwencji wszyscy jego podwładni.
- Problem eliminuje zamiana wiersza `WHERE E.department_ID = D.department_ID` na `WHERE E.department_ID = D.department_ID(+)`

```

-- Poprawny wynik zapytania

EMPLOYEE_ID MANAGER_ID NAME                DEPARTMENT_NAME          MAN-
AGER_ID
-----
-
          204          101 ...Hermann Baer   Public Relations          204
          205          101 ...Shelley Higgins Accounting                205
          206          205 .....William Gietz Accounting                205
          203          101 ...Susan Mavris   Human Resources          203
          201          100 ..Michael Hartstein Marketing                201
          202          201 ....Pat Fay       Marketing                201

6 rows selected

```

Przykład 11b – zapytania hierarchiczne

```

-- Składnia dotychczasowa

SELECT
  E.employee_ID,
  E.manager_ID,
  RPAD( RPAD(' ', 2*(LEVEL)-1, '.')||E.first_name ||' '||E.last_name, 30) Name,
  D.department_name,
  D.manager_ID
FROM
  Employees E, Departments D
WHERE
  E.department_ID = D.department_ID(+) AND
  D.manager_ID > 200
START WITH
  E.employee_ID = 100
CONNECT BY
  E.manager_ID = PRIOR E.employee_ID;

EMPLOYEE_ID MANAGER_ID NAME                DEPARTMENT_NAME          MANAGER_ID
-----

```

204	101	...Hermann Baer	Public Relations	204
205	101	...Shelley Higgins	Accounting	205
206	205William Gietz	Accounting	205
203	101	...Susan Mavris	Human Resources	203
201	100	..Michael Hartstein	Marketing	201
202	201	...Pat Fay	Marketing	201

6 rows selected

```
-- Składnia ANSI
SELECT
  E.employee_ID,
  E.manager_ID,
  RPAD( RPAD(' ', 2*(LEVEL)-1, '.')||E.first_name ||' '||E.last_name, 30) Name,
  D.department_name,
  D.manager_ID
FROM
  Employees E LEFT OUTER JOIN Departments D ON E.department_ID = D.department_ID
WHERE
  D.manager_ID > 200
START WITH
  E.employee_ID = 100
CONNECT BY
  E.manager_ID = PRIOR E.employee_ID;
```

EMPLOYEE_ID	MANAGER_ID	NAME	DEPARTMENT_NAME	MANAGER_ID
203	101	...Susan Mavris	Human Resources	203
204	101	...Hermann Baer	Public Relations	204
205	101	...Shelley Higgins	Accounting	205
206	205William Gietz	Accounting	205
201	100	..Michael Hartstein	Marketing	201
202	201	...Pat Fay	Marketing	201

6 rows selected

Komentarz:

- Sortowanie zapytań hierarchicznych, ze względu na konieczność zachowania podległości, nie jest bezpośrednio możliwe⁷. Niestety analogiczne jak wydawałoby się zapytanie w notacji ANSI, zwraca wynik inaczej posortowany. Na szczęście hierarchia jest zachowana poprawnie.

5. Wnioski

W artykule pokazano, że z pozoru proste zagadnienia jakim są połączenia relacyjne, po uwzględnieniu wielu zagadnień szczegółowych, stają się podatne na błędy. Jeśli dodatkowo uwzględnimy możliwość korzystania z nowej składni ANSI, ewentualnych problemów przybędzie.

Składnia ANSI SQL nie zawsze jest w 100% odpowiednikiem składni dotychczasowej. Warto o tym zawsze pamiętać.

Dużo miejsca poświęcono zagadnieniu istnienia w tabelach kolumn NULL. Pokazano, że jest to często bardzo niebezpieczne. Łatwo wyobrazić sobie sytuację, gdy z „dobrodziejstwa” nie wpisania wartości do kolumny skorzystamy długo po napisaniu np. aplikacji raportującej. Jeśli na etapie jej powstawania nie uwzględnimy tego faktu, to któregoś dnia raport po prostu wyprodukuje błędne / mylące wyniki. Innymi słowy dopiero po jakimś czasie, na skutek modyfikacji danych, z których korzysta zapytanie, pojawi się błąd.

Bibliografia

⁷ Oracle 9i wprowadza jednak zmodyfikowaną klauzulę ORDER SIBLING BY, która sortuje wyniki z zachowaniem hierarchii.

- [HRSchema] Oracle® Database, Sample Schemas, 10g Release 1 (10.1), Part No. B10771-01
- [Genni00] Gennick J.: An Incremental Approach to Developing SQL Queries. Oracle Magazine, July/August 2000, Volume XIV, Issue 4
- [Czupr03] Czuprynski J.: Getting ANSI About Joins, dostępne na otn.oracle.com
- [ORASQL] Oracle® Database SQL Reference 10g Release 1 (10.1) Part Number B10759-01