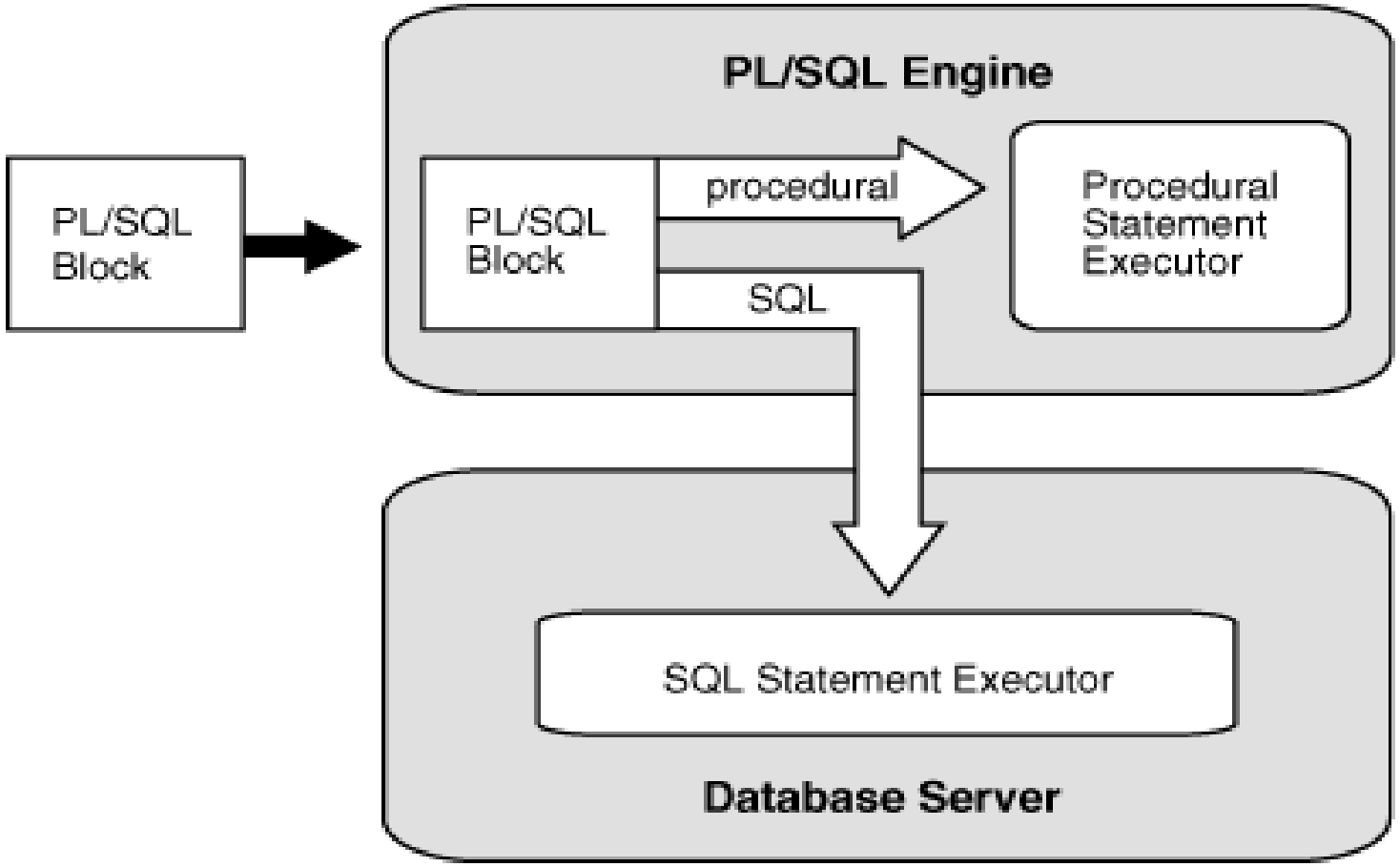
PL/SQL architektúra



A PL/SQL motor

- A PL/SQL nyelvű egységek fordítását és futtatását végző rendszer
  - az adatbázis-kezelőben
  - fejlesztőeszközben (pl. Oracle Forms)
- Ha nincs benne SQL-utasítás, a teljes egység feldolgozását ő végzi, különben az SQL-utasítás(oka)t továbbítja az SQL motornak

Fordítási egységek

- Blokk
- Tárolt függvény
- Tárolt eljárás
- Csomag specifikáció
- Csomag törzs
- Trigger
- Típus specifikácó
- Típus törzs

Programfejlesztés PL/SQL-ben

- SQL*Plus / *i*SQL*Plus
- SQLDeveloper
- JDeveloper
- Oracle Forms Developer
- Oracle Reports
- …

PL/SQL programok végrehajtása

- Szerver oldalon (Oracle DBMS)
- Eszköz (kliens) oldalon (pl. Oracle Forms Developer)
- (3GL nyelv előfordítóval)

Szerver oldal	**Eszköz oldal**
Névtelen blokk	Névtelen blokk
Tárolt alprogramok	Alkalmazás alprogramjai
Tárolt csomagok	Alkalmazás csomagjai
Adatbázistriggerek	Alkalmazástriggerek
Objektumtípusok	Objektumtípusok

Függőségek kezelése

- Néhány sémaobjektum-fajta definícójában más objektumokra hivatkozhat
  - pl. nézet táblákra és más nézetekre, alprogramtörzs más alprogramokra ill. táblákra és nézetekre, stb.
- Ha egy A objektum hivatkozik egy B objektumra, akkor
  - A (B-től) függő
  - B-t (A által) hivatkozott

  objektumnak nevezzük.
- Ha módosítjuk a hivatkozott objektumot, a függő objektumok hibássá válhatnak (pl. eldobjuk a nézet alapjául szolgáló táblák valamelyikét)

Függőségek felderítése

- USER_DEPENDENCIES, ALL_DEPENDENCIES és DBA_DEPENDENCIES statikus adatszótárnézetek segítségével

ALL_DEPENDENCIES

OWNER	VARCHAR2(30)	NOT NULL	Az obj. tulajdonosa
NAME	VARCHAR2(30)	NOT NULL	Az obj. neve
TYPE	VARCHAR2(17)		Az ob. típusa
REFERENCED_OWNER	VARCHAR2(30)		A hiv. obj. tulajdonosa
REFERENCED_NAME	VARCHAR2(64)		A hiv. obj. neve
REFERENCED_TYPE	VARCHAR2(17)		A hiv. obj. típusa
REFERENCED_LINK_NAME	VARCHAR2(128)		Az adatb. kapcsolat neve (ha távoli)
SCHEMAID	NUMBER		Sémaazonosító
DEPENDENCY_TYPE	VARCHAR2(4)		REF vagy HARD

Objektumok állapota

- Érvényes (VALID)
- Fordítási hibás (Compiled with errors)
- Érvénytelen (INVALID)
- Jogosulatlan (Unauthorized)

Érvénytelenítődés

- Ha A függ B-től, B pedig C-től, akkor A közvetlenül függ B-től és közvetetten C-től
- A közvetlenül függő objektumok csak akkor érvénytelenítődnek, ha a hivatkozott objektum változása (a hiv. obj. szignatúrájáé) érinti őket
- A közvetett függőségek mentén érvénytelenítődhet, ha őt magát nem is befolyásolja (kaszkádolt érvénytelenítés)
  - pl. ha C megváltozása érvényteleníti B-t, akkor A-t is

Újraérvényesítés

- Automatikusan megy
  - pl. újrafordítással

Névfeloldás

- Fordításkor a fordító határozza meg, hogy egy programegység neveivel mely objektumok állnak kapcsolatban
- Egy név hivatkozhat lokális változóra, táblára, csomagra, alprogramra, sémára, stb.

```
CREATE OR REPLACE PACKAGE pkg1 AS
  m NUMBER;
  TYPE t1 IS RECORD (a NUMBER);
  v1 t1;
  TYPE t2 IS TABLE OF t1 INDEX BY PLS_INTEGER;
  v2 t2;
  FUNCTION f1 (p1 NUMBER) RETURN t1;
  FUNCTION f2 (q1 NUMBER) RETURN t2;
END pkg1;
/
CREATE OR REPLACE PACKAGE BODY pkg1 AS
  FUNCTION f1 (p1 NUMBER) RETURN t1 IS
    n NUMBER;
  BEGIN
    -- (1) minősítetlen név
    n := m;
    -- (2) ponttal elválasztott névlista
    n := pkg1.m;
    -- (3) ponttal elválasztott névlista
    n := pkg1.f1.p1;
    -- (4) ponttal elválasztott névlista
    n := v1.a;
    -- (5) ponttal elválasztott névlista
    n := pkg1.v1.a;
    -- (6) indexelt név mezőnév-hivatkozással
    n := v2(10).a;
    -- (7) függvényhívás mezőnév-hivatkozással
    n := f1(10).a;
    -- (8) függvényhívás indexeléssel,
    -- majd mezőnév-hivatkozással
    n := f2(10)(10).a;
    -- (9) függvényhívás (ami egy ponttal
    -- elhatárolt névlista) indexeléssel,
    -- majd mezőnév-hivatkozással
    n := hr.pkg1.f2(10)(10).a;
-- (10) változónév mezőnév-
    -- hivatkozással
    v1.a := p1;
    RETURN v1;
  END f1;

  FUNCTION f2 (q1 NUMBER) RETURN t2 IS
    v_t1 t1;
    v_t2 t2;
  BEGIN
    v_t1.a := q1;
    v_t2(1) := v_t1;
    RETURN v_t2;
  END f2;
END pkg1;
/
```

Névfeloldás (SQL)

Az objektumnevek pontokkal elválasztott részekből állnak, pl.: `HR.EMPLOYEES.SALARY`

1. Az első részt értelmezi először:
   a) Az aktuális sémában keresi az objektumot az adott név első részével. Ha nem talál ilyet, akkor b)
   b) Egy publikus szinonímát keres, amely megfelel a név első részének. Ha nem talál, akkor c)
   c) Keres egy sémát, amelynek a neve megegyezik a név első részével. Ha talál ilyet, akkor a a) lépést használjuk a név második részével. Ha nem talál ilyet, akkor hibát kapunk.
2. Ha megtaláltuk az első részhez tartozó objektumot, akkor az objektumnév maradék részei a megtalált objuktum érvényes részeinek kell lennie.

Névfeloldás (PL/SQL)

- Ha egy SQL utasításban egy olyan nevet használunk, amely egy oszlophoz is és egy változóhoz is tartozik, akkor ott az oszlopnév lesz értelmezve.

Névfeloldás (PL/SQL)

```
CREATE TABLE employees2 AS
 SELECT LAST_NAME FROM employees;
DECLARE
  last_name VARCHAR2(10) := 'King';
BEGIN
 DELETE FROM employees2
 WHERE LAST_NAME = last_name;
 DBMS_OUTPUT.PUT_LINE('Deleted '
   || SQL%ROWCOUNT || ' rows.');
END;
/

Result:
Deleted 107 rows.
```

```
DECLARE
  v_last_name VARCHAR2(10) := 'King';
BEGIN
 DELETE FROM employees2
  WHERE LAST_NAME = v_last_name;
 DBMS_OUTPUT.PUT_LINE('Deleted '||SQL%ROWCOUNT||' rows.');
END;
/
Result: Deleted 2 rows.

<<main>>
DECLARE
 last_name VARCHAR2(10) := 'King';
BEGIN
 DELETE FROM employees2
  WHERE last_name = main.last_name;
 DBMS_OUTPUT.PUT_LINE('Deleted '||SQL%ROWCOUNT||' rows.');
END;
/
Result: Deleted 2 rows.
```

```
DECLARE
  FUNCTION dept_name (department_id IN NUMBER)
    RETURN departments.department_name%TYPE IS
  department_name departments.department_name%TYPE;
BEGIN
  SELECT department_name INTO dept_name.department_name
  -- ^column                    ^local variable
  FROM departments
  WHERE department_id = dept_name.department_id;
      -- ^column          ^formal parameter
  RETURN department_name;
END dept_name;
BEGIN
 FOR item IN (SELECT department_id FROM departments) LOOP
   DBMS_OUTPUT.PUT_LINE ('Department: ' ||
   dept_name(item.department_id));
 END LOOP;
END;
/
```

SQL vs. PL/SQL

- A PL/SQL és az SQL névfeloldási szabályai hasonlók.
- A PL/SQL ugyanazt a névfeloldási mechanizmust használja, mint az SQL, az SQL utasítások feldolgozásakor
  - pl. HR.JOBS -> HR sémabeli csomagok, típusok, táblák és nézetek között keres
- PL/SQL utasításban (pl. értékadás, alprogramhívás) eltérő a sorrend!
  - HR.JOBS -> először az aktuális séma csomagjai, típusai, táblái és nézetei között keres, majd csak ezután a HR sémában

Névfeloldás PL/SQL-beli statikus SQL utasításokban

Ha a PL/SQL fordító egy statikus SQL utasítást talál:

1. Ha az utasítás SELECT, akkor eltávolítja az INTO részt
2. Elküldi az utasítást az SQL motornak
3. Az SQL motor ellenőrzi az utasítás szintaktikáját
4. Ha a szintaktika nem helyes, a PL/SQL egység fordítása nem sikerül. Ha szintaktika helyes, akkor az SQL motor meghatározza a táblák neveit, és megpróbálja feloldani az utasításban lévő többi nevet is
5. Ha az SQL motor nem tud egy nevet feloldani, akkor a nevet visszaküldi a PL/SQL fordítónak. Az ilyen nevet úgy hívják, hogy „escaped identifier”.
6. A PL/SQL fordító megpróbálja feloldani az „escaped identifier”-t.

…

Névfeloldás PL/SQL-beli statikus SQL utasításokban

7. Először a PL/SQL egységen belül próbálja feloldani. Ha nem sikerül, akkor séma szinten próbálja feloldani, Ha az sem sikerül, akkor a PL/SQL egység fordítása nem sikerül.
8. Ha a PL/SQL egység fordítása sikeres, akkor a PL/SQL fordító az eredetivel megegyező, szabályos SQL utasításszöveget generál, és ezt tárolja a generált programkóddal.
9. Futtatási időben a PL/SQL futtató rendszer olyan rutinokat hív meg, amely elemzi, összerendeli, és futtatja a szabályos SQL utasítást.
10. A hozzárendelt argumentumok az „escaped identifier”-ok.
11. Ha az utasítás egy SELECT utasítás, a PL/SQL futtató rendszer az eredményt az INTO-ban meghatározott PL/SQL célváltozóban tárolja. (Amelyet az 1-es pontban eltávolítottunk.)

```
BEGIN
 <<block1>>
 DECLARE
 TYPE Client IS RECORD
      ( first_name VARCHAR2(20),
        last_name VARCHAR2(25) );
 TYPE Customer IS RECORD
      ( first_name VARCHAR2(20),
        last_name VARCHAR2(25) );
 BEGIN
  DECLARE
  client Customer; -- declaration of variable client
  lead block1.Client; -- qualified reference to type Client
    BEGIN
    NULL;
    END;
  END;
END;
/
```

Capture

- Outer capture
- Same-scope capture
- Inner capture: Ha egy belső hatásköri egység neve nem hivatkozik már egy külső hatásköri egység egyedére
  - vagy a belső egység egyede
  - vagy hiba

Inner capture, példa

```
CREATE TABLE tab1 (col1 NUMBER, col2
  NUMBER);
INSERT INTO tab1 VALUES (100, 10);
CREATE TABLE tab2 (col1 NUMBER);
INSERT INTO tab2 VALUES (100);
CREATE OR REPLACE PROCEDURE proc AS
   CURSOR c1 IS SELECT * FROM tab1
  WHERE EXISTS (SELECT * FROM tab2
  WHERE col2 = 10);
BEGIN NULL; END;
/
ALTER TABLE tab2 ADD (col2 NUMBER);
```

Példa

```
declare
manager_id number(5):=100;
v_name varchar2(50);
begin
select first_name||last_name
into v_name
from hr.employees
where employee_id=manager_id;
dbms_output.put_line('v_name');
end;
/
```

Hogyan kerüljük el az inner capture problémát Select és DML utasításokban?

- Adjunk meg egyedi aliasokat az utasításokban
- Ne használjunk olyan aliast, amely egy olyan sémabeli név, amelynek van olyan objektuma, amelyet az utasítás hivatkozik
- Minden oszlophivatkozást minősítsünk a megfelelő tábla aliasával

```
CREATE OR REPLACE TYPE type1  AS OBJECT (a NUMBER); /
DROP TABLE tab1;
CREATE TABLE tab1 (tab2 type1);
INSERT INTO tab1 (tab2)
VALUES (type1(10));
DROP TABLE tab2;
CREATE TABLE tab2 (x NUMBER);
INSERT INTO tab2 (x) VALUES (10);
SELECT * FROM tab1 hr
   WHERE EXISTS (SELECT * FROM hr.tab2
                 WHERE x = hr.tab2.a);
ALTER TABLE tab2 ADD (a NUMBER);

SELECT * FROM hr.tab1 p1
    WHERE EXISTS (SELECT * FROM hr.tab2 p2
                  WHERE p2.x = p1.tab2.a);
```

SQL adatbázis-objektum nevek

- Egy névtéren belül két objektumnak nem lehet ugyanaz a neve.
- Egy névtéren osztoznak a következő sémaobjektumok: táblák, nézetek, szekvenciák, privát szinonimák, tárolt alprogramok, csomagok, materializált nézetek, felhasználó által definiált típusok
- A következő sémaobjektumoknak saját névterük van: indexek, megszorítások, klaszterek, triggerek, privát adatbázislinkek, dimenziók

SQL adatbázis-objektum nevek

- A következő nem sémabeli objektumoknak saját névterük van: szerepkörök, publikus szinonimák, publikus adatbázislinkek, táblaterek, …
- Egy tábla vagy nézet oszlopnevei nem lehetnek azonosak, de különböző táblák vagy nézetek oszlopnevei igen.
- Egy csomag alprogramjainak a nevei lehetnek ugyanazok, hogyha az argumentumaik száma vagy típusa nem egyezik meg. (Túlterhelés)

Módosítható nézetek

Módosítható nézetek nem tartalmazhatják a következőket:

- Halmazműveletek (INTERSECT, MINUS, UNION, UNION ALL)
- DISTINCT
- Group Aggregate Functions (AVG, COUNT, MAX, MIN, SUM, etc.)
- GROUP BY
- ORDER BY
- CONNECT BY
- START WITH
- Kollekciókifejezés a select listában
- Allekérdezés a select listában
- Táblák összekapcsolása (kivéve néhány kivétel)

Ha pszeudooszlopok vannak a nézetben, akkor azok nem lehetnek update utasításban. (COLUMN_VALUE, ROWID, ROWNUM,)

```
CREATE TABLE demo_tab (person_id NUMBER(3),
                       first_name VARCHAR2(20),
                       last_name VARCHAR2(20));
CREATE OR REPLACE VIEW upd_view AS SELECT * FROM demo_tab;
INSERT INTO demo_tab (person_id, first_name, last_name)
   VALUES (1, 'Daniel', 'Morgan');
INSERT INTO demo_tab (person_id, first_name, last_name)
   VALUES (2, 'Helen', 'Lofstrom');
COMMIT;
SELECT * FROM upd_view;
UPDATE upd_view SET person_id = person_id * 10;
SELECT * FROM upd_view;
desc user_updatable_columns

SELECT table_name, column_name, updatable, insertable, deletable
  FROM user_updatable_columns
  WHERE table_name IN ( SELECT view_name FROM user_views);
```

Nem módosítható nézetek

```
CREATE OR REPLACE VIEW nonupd_view AS
   SELECT DISTINCT *
   FROM demo_tab;

   SELECT table_name, column_name, updatable, insertable,
   deletable
   FROM user_updatable_columns
   WHERE table_name = 'NONUPD_VIEW';

   SELECT * FROM nonupd_view;

   UPDATE nonupd_view
   SET person_id = person_id * 10;
```

```
CREATE VIEW EUR_ORSZ AS
  SELECT azon, orszag FROM s_orszagok WHERE
   foldresz='Európa';

insert into eur_orsz values (11111,‘Csodaország');

CREATE OR REPLACE TRIGGER INSTED INSTEAD OF
  INSERT ON eur_orsz FOR EACH row BEGIN
  INSERT INTO s_orszagok (azon, orszag, foldresz)
    VALUES (:new.azon,:new.orszag,'Európa’);
END;
/
insert into eur_orsz values (22222,'Meseország');
```

További példák:

http://psoug.org/reference/instead_of_trigger.html

```
CREATE TABLE test (testcol VARCHAR2(15));
INSERT INTO test VALUES ('dummy');
CREATE OR REPLACE TRIGGER follows_a
   AFTER UPDATE ON test FOR EACH ROW
   BEGIN dbms_output.put_line('A');
   END follows_a;
   /
CREATE OR REPLACE TRIGGER follows_b
   AFTER UPDATE ON test FOR EACH ROW
   BEGIN dbms_output.put_line('B');
   END follows_b;
   /
set serveroutput on
UPDATE test SET testcol = 'a';
CREATE OR REPLACE TRIGGER follows_b
   AFTER UPDATE ON test FOR EACH ROW
   FOLLOWS follows_a
   BEGIN dbms_output.put_line('B');
   END follows_b;
   /
UPDATE test SET testcol = 'a';
```

```
DROP TABLE log;
CREATE TABLE log ( log_id NUMBER(6),
                   up_date DATE,
                   new_sal NUMBER(8,2),
                   old_sal NUMBER(8,2) );
-- Autonomous trigger on employees table:
CREATE OR REPLACE TRIGGER log_sal
   BEFORE UPDATE OF salary ON employees
   FOR EACH ROW
 DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
 BEGIN
  INSERT INTO log ( log_id, up_date, new_sal, old_sal )
   VALUES (:old.employee_id,SYSDATE,:new.salary,:old.salary);
  COMMIT;
 END;
/
```

```
UPDATE employees
  SET salary = salary * 1.05
  WHERE employee_id = 115;
COMMIT;
UPDATE employees
  SET salary = salary * 1.05
  WHERE employee_id = 116;
ROLLBACK;
-- Show that both committed and rolled-back updates
-- add rows to log table
SELECT *
FROM log
WHERE log_id = 115 OR log_id = 116;
Result:
LOG_ID     UP_DATE   NEW_SAL    OLD_SAL
---------- --------- ---------- ----------
115        25-AUG-09 3255       3100
116        25-AUG-09 3045       2900
2 rows selected.
```

```
conn / as sysdba
CREATE USER abc IDENTIFIED BY abc DEFAULT TABLESPACE uwdata
TEMPORARY TABLESPACE temp QUOTA 10M ON uwdata;
GRANT create session, create table, create procedure to abc;
conn abc/abc
CREATE TABLE t1 (mycol VARCHAR2(20));
CREATE TABLE t2 (yourcol NUMBER(10,2));
CREATE TABLE t3 (ourcol DATE);

conn uwclass/uwclass
CREATE OR REPLACE PROCEDURE definer_test AUTHID DEFINER IS
BEGIN
 FOR rec IN (SELECT table_name FROM user_tables)
   LOOP
       dbms_output.put_line(rec.table_name);
    END LOOP;
   END definer_test;
   /
```

```
CREATE OR REPLACE PROCEDURE cu_test AUTHID CURRENT_USER IS
   BEGIN
     FOR rec IN (SELECT table_name FROM user_tables)
     LOOP
       dbms_output.put_line(rec.table_name);
     END LOOP;
   END cu_test;
   /

   set serveroutput on
   exec definer_test;
   exec cu_test;
   GRANT execute on definer_test TO abc;
   GRANT execute on cu_test TO abc;

   conn abc/abc
   set serveroutput on
   exec uwclass.definer_test;
   exec uwclass.cu_test;
```

```
CREATE TABLE cascade (
testcol VARCHAR2(10));

CREATE OR REPLACE TRIGGER t_cascade
AFTER INSERT
ON cascade

BEGIN
  INSERT INTO cascade
  (testcol)
  VALUES
  ('change');
END t_cascade;
/

INSERT INTO cascade (testcol) VALUES ('ABC');
```

With utasításrész

- Egyszeres másodlagos név:

```
WITH <alias_name> AS (subquery_sql_statement)
 SELECT <column_name_list> FROM <alias>;
```

Pl:
```
WITH q AS (SELECT dummy FROM dual)
 SELECT dummy FROM q;
```

- Többszörös másodlagos név:

```
WITH <alias_one> AS (subquery_sql_statement),
     <alias_two> AS (sql_statement_from_alias_one)
SELECT <column_name_list>
 FROM <alias_one>, <alias_two>
 WHERE <join_condition>;
```

```
WITH qb1 AS
     (SELECT inst_id FROM gv$session),
    qb2 AS
     (SELECT unique inst_id FROM qb1
      UNION ALL
      SELECT unique inst_id FROM qb1)
   SELECT /*+ MATERIALIZE */ *
   FROM qb1, qb2
   WHERE qb1.inst_id = qb2.inst_id;
```

```
SELECT per_h_email FROM person p
   WHERE p.per_ok2_email = 'A'
   AND p.per_h_email IN ( SELECT person_id
                          FROM person
                          WHERE per_fmw = 'Y')
   UNION
   SELECT po_w_email FROM poie o
   WHERE o.po_status = 'A'
   AND o.po_w_email IN ( SELECT person_id
                         FROM person
                         WHERE per_fmw = 'Y');
```

Helyette:

```
WITH w AS ( SELECT person_id FROM person WHERE per_fmw = 'Y')
   SELECT per_h_email
   FROM person p, w
   WHERE p.person_id = w.person_id
   AND p.per_ok2_email = 'A'
   AND p.per_h_email IS NOT NULL
   UNION
   SELECT po_w_email FROM poie o, w
   WHERE o.person_id = w.person_id
   AND o.po_status = 'A'
   AND o.po_w_email IS NOT NULL;
```