IDSe Demo 3 – Advanced Oracle SQL Queries

Marek Rychlý, rychly@fit.vutbr.cz

Outline / Table of Content

The queries in this lecture will utilise the following tables:

```
DROP TABLE book_author;
DROP TABLE author;
DROP TABLE book;
```

```
DROP TABLE category;

CREATE TABLE category (
        category_id NUMBER(8) PRIMARY KEY,
        category_name VARCHAR2(32) NOT NULL
);
CREATE TABLE book (
        isbn NUMBER(13) PRIMARY KEY,
        title VARCHAR2(128) NOT NULL,
        category_id REFERENCES category(category_id)
);
CREATE TABLE author (
        id NUMBER(8) PRIMARY KEY,
        family_name VARCHAR2(32) NOT NULL,
        given_names VARCHAR2(32),
        born DATE
);
CREATE TABLE book_author (
        book_isbn REFERENCES book(isbn),
        author_id REFERENCES author(id),
        PRIMARY KEY (book_isbn, author_id)
);
INSERT INTO author VALUES (1, 'Windham', 'John', DATE '1903-06-10');
INSERT INTO author VALUES (2, 'Steinbeck', 'John', DATE '1902-02-27');
INSERT INTO author VALUES (3, 'Kerouac', 'Jack', DATE '1922-03-12');
```

Retrieving Data Using the SQL SELECT Statement

1. A SELECT statement retrieves data from database. See example:

```
SELECT title, isbn FROM book WHERE title LIKE 'Handbook of %';
```

   will retrieve titles and isbn numbers of all books which title starts with "Handbook of ...". In this SELECT statements

   - `FROM book` part represents a relation,
   - `SELECT title, isbn` part represents a projection operation on the relation,
   - and `WHERE title LIKE 'Handbook of %'` part represents a selection operation on the relation

2. The SELECT can retrieve multiple rows with the same contents, for example,

```
SELECT given_names FROM author;
```

   can retrieve name "John" several-times if there are multiple authors with their first names "John", e.g., John Wyndham and John Steinbeck. In this case, the SELECT can be modified to retrieve just unique rows:

```
SELECT DISTINCT given_names FROM author;
```

3. For more examples of queries, see Oracle Programming/Retrieving Data Using the SQL SELECT Statement. For more details on the SELECT statement, see Oracle Database SQL Language Reference: About Queries and Subqueries.

Retrieving Ordered Data Using the ORDER BY Clause

1. Data retrieved by a SELECT statement can be ordered. See example:

```
SELECT family_name, given_names FROM author ORDER BY family_name ASC, given_names ASC;
```

   which will list all authors ascending (alphabetical) order by their full names. You can use keywork "DESC" for descending order.

2. Each row of data retrieved by every SELECT statement has two (hidden) system columns, namely ROWNUM and ROWID, which are, for each row, set to its number (the first has number 1, the second has number 2, etc.) and its unique identifier, respectively. The column ROWNUM and the ORDER BY clause can be utilised together to retrieve rows with maximal or minimal values, first X rows, etc. For example,

```
3. SELECT * FROM (
4.   SELECT DISTINCT family_name FROM author ORDER BY family_name DESC
```

```
) WHERE ROWNUM <= 2;
```

retrieves family names of the last two authors at the end of the alphabet (i.e., the last two in the list of authors ordered by their family names). Please, note that in the case of rows with maximal or minimal values, there is another way how to query the same results by aggregation function MAX, instead of ordering the results in combination with ROWNUM.

5. For more examples of queries, see Oracle Programming/Restricting and Sorting Data.

Retrieving Aggregated Data Using the Group Functions

1. Records retrieved by the SELECT statements can be grouped into groups with identical values of their columns (e.g., in grouping of authors by their first names, there will be group of all autors with name "John", another group of all authors with name "Jack", etc.). Then, records in each group can be aggregated by a (group) function to get count of grouped records in each group, a record with maximal or minimal values, etc. (e.g., for the example above, we can get how many auhtor with names "Jonh" or "Jack" are there). See example:

```
SELECT given_names, count(*) AS number_of_authors FROM author GROUP BY given_names;
```

2. Note that the projection part of SELECT statements with grouping can contain only columns used in the GROUP BY calues and/or aggregation functions, i.e., the projection can show only those values which are identical for the whole group of records In the example above, given names all identical for are authors in the group (because the group contains all authors with the same givens names) and count is an aggregation function applied on (or a property of) the whole group.
3. We can also select particular rows from the result of the grouping (a SELECT statement with the GROUP BY clause) by the HAVING clause. See example:

```
4. SELECT given_names, count(*) AS number_of_authors FROM author WHERE given_names LIKE 'J%'
   GROUP BY given_names HAVING count(*) >= 2;
```

5. For more details, see Oracle Database SQL Language Reference: Aggregate Functions.

Combining Records from Multiple Tables Using the SQL JOIN Clause

SQL queries can retrieve data from combination of multiple tables by their joins. There are three types of joins: Natural Join, Inner Join, and Outer Join.

For more details and examples, see Oracle Programming/10g Advanced SQL.

Natural Join

1. The NATURAL JOIN joins two tables which contain a column or multiple columns with the same name and data-type. For example,

```
SELECT isbn, title, category_name FROM book NATURAL JOIN category;
```

   will list for each book its isbn number, title, and its category name.

2. It is recommended to use the Inner Join, which explicitely describes columns in the join condition, rather than Natural Join. The reason is that future altering of tables (e.g., renaming its columns) may break natural join using the tables.

Inner Join

1. The INNER JOIN joins two or more tables, returning only the rows that satisfy the JOIN condition. For example,

```
2. SELECT a.family_name, a.given_names, b.title
3. FROM author a
4.   INNER JOIN book_author ba ON (a.id = ba.author_id)
  INNER JOIN book b ON (ba.book_isbn = b.isbn);
```

will list all pairs of an author and a book such that the book is written by the author.

5. Keyword "INNER" is oprional. Also note that the following query is an alternative to the example query from the previous section dealing with Natural Join. The query

```
SELECT isbn, title, category_name
FROM book b JOIN category c ON (b.category_id = c.category_id);
```

will list for each book its isbn number, title, and its category name.

Outer Join

1. The OUTER JOIN joins two or more tables, returning all values whether or not the join condition is met. When a value exists in one table but not the other, nulls are used in the place of the columns that are joined to a record without a JOIN companion.
2. There are three types of outer joins: Full Outer Join, Left Outer Join, and Right Outer Join.
    - With the FULL OUTER JOIN the query will return rows from either of the tables joined, whether or not there is any matching data on the table joined. If no matching data exists, nulls are placed into the fields where data would have otherwise existed. For example,

```
    - SELECT a.family_name, a.given_names, b.title
```

```
FROM author a
    FULL OUTER JOIN book_author ba ON (a.id = ba.author_id)
    FULL OUTER JOIN book b ON (ba.book_isbn = b.isbn);
```

will list all pairs of an author and a book such that the book is written by the author. Contrary to the example query in the previous section dealing with Inner Join, **the query above will list all authors and all books, even the authors without any book written and the books without any author (e.g., The Bible)**.

- With the LEFT OUTER JOIN the query will return rows only if the row exists in the table specified *on the left side* of the join. When no matching data is found from the table *on the right side* of the join, nulls are placed into the fields where the data would have otherwise existed. For example,

```
SELECT a.family_name, a.given_names, b.title
FROM author a
    LEFT OUTER JOIN book_author ba ON (a.id = ba.author_id)
    LEFT OUTER JOIN book b ON (ba.book_isbn = b.isbn);
```

will list all pairs of an author and a book such that the book is written by the author. Contrary to the example query in the previous section dealing with Inner Join, **the query above will list even the authors without any book written**.

- With the RIGHT OUTER JOIN the query will return rows only if the row exists in the table specified *on the right side* of the join. When no matching data is found from the table *on the left side* of the join, nulls are placed into the fields where the data would have otherwise existed. For example,

```
SELECT a.family_name, a.given_names, b.title
FROM author a
   RIGHT OUTER JOIN book_author ba ON (a.id = ba.author_id)
   RIGHT OUTER JOIN book b ON (ba.book_isbn = b.isbn);
```

will list all pairs of an author and a book such that the book is written by the author. Contrary to the example query in the previous section dealing with Inner Join, **the query above will list even the books without any author (e.g., The Bible)**.

Using Subqueries to Solve Queries

1. There are two ways how to get just those records with maximum values from all records from table(s). See example:

```
2. SELECT given_names, family_name FROM author WHERE family_name = (SELECT MAX(family_name) FROM author);
3. SELECT given_names, family_name FROM author a1 WHERE NOT EXISTS (
4.   SELECT 1 FROM author a2 WHERE a2.family_name > a1.family_name
```

```
);
```

The both queries above will return the authors who are the last in the alphabet according to their family names. The second query let you be more precise, e.g., let you to ignore some authors in the condition of maximality (you can try to select the authors who are the last in the alphabet according to their family names but who does not have first name "Jack" etc.).

Practice Exercise

1. Run the Oracle SQL Developer and connect to the Oracle database at BUT FIT.
2. Create the tables defined in this lecture and insert sample data into the tables (sample book, authors, and categories). Run the queries described in this lecture and evaluate the resulting data.
3. Feel free to experiment with running another SELECT statements in the Oracle SQL Developer.

IDSe Demo 4 – Database Triggers, Stored Procedures, and Indexing in Oracle Database

Marek Rychlý, rychly@fit.vutbr.cz

Outline / Table of Content

The queries in this lecture will utilise the same tables as they were defined in the previous lecture. Moreover, we add the following database objects:

```
DROP SEQUENCE category_seq;
DROP SEQUENCE author_seq;
```

```
CREATE SEQUENCE category_seq;
CREATE SEQUENCE author_seq START WITH 4;

INSERT INTO author VALUES (author_seq.nextval, 'Heinlein', 'Robert Anson', DATE '1907-07-07');
```

PL/SQL Language

1. The Procedural Language extension of SQL (PL/SQL) is a combination of SQL and the procedural features of programming languages.
2. Each PL/SQL program is a sequence of SQL and PL/SQL statements in a PL/SQL block. The PL/SQL block ends with the reserved keyword END and consists of three parts:
    - declaration section (optional; usualy starts with the reserved keyword DECLARE) – to declare variables, constants, records and cursors, which are used to manipulate data in the execution section,
    - execution section (mandatory; starts with the reserved keyword BEGIN) – to define a program logic performing a given task,
    - exception (error) handling section (optional; starts with the reserved keyword EXCEPTION) – to catch any errors in the program, so that the PL/SQL block dows not terminate bruptly with errors.

3. PL/SQL blocks may have the following structure:

```
4. DECLARE
5.   ---- declarations
6.   -- a named variable of a given datatype and optionaly with a particular value
7.   variable_name datatype [NOT NULL := value];
8.   -- a named constant of a given datatype and value
9.   constant_name CONSTANT datatype := value;
10.  -- a named variable of the same datatype as a datatype of a given column of a given table
11.  column_variable_name table_name.column_name%TYPE;
12.  -- a named variable which will contain a record (a row) from a given table
13.  row_record_variable_name table_name%ROWTYPE;
14.  -- a named cursor which will be used to read the result of a query defined by a given select statement
15.  CURSOR cursor_name IS select_statement;
16.  -- a named exception representing Oracle error with a given ORA code
17.  exception_name EXCEPTION; PRAGMA EXCEPTION_INIT(exception_name, err_code);
18.BEGIN
19.  ---- a program logic (a source code of the program)
20.  statements;
21.  -- conditional statements
22.  IF condition THEN
23.          statement_1;
24.  ELSE
25.          statement_2;
26.  END IF;
27.  -- iterative statements
28.  LOOP
29.          statements;
```

```
30.         EXIT WHEN condition;
31.  END LOOP;
32.  WHILE condition LOOP
33.         statements;
34.  END LOOP;
35.  FOR counter IN val1..val2 LOOP
36.         statements;
37.  END LOOP;
38.  -- cursor loops
39.  FOR new_row_record_variable_name IN cursor_name LOOP
40.         statements which can access new_row_record_variable_name.column_name and others;
41.  END LOOP;
42.  -- SQL queries
43.  SELECT column INTO variable FROM ...;
44.  IF SQL%NOTFOUND ...
45.  IF SQL%FOUND ...
46.  IF SQL%ROWCOUNT ...
47.  -- SQL insert statements
48.  INSERT ...;
49.  -- SQL update/delete statements and conditional statements on affected rows
50.  UPDATE ...;
51.  DELETE ...;
52.  IF SQL%NOTFOUND ...
53.  IF SQL%FOUND ...
54.  IF SQL%ROWCOUNT ...
55.EXCEPTION
56.  -- exception handlers
57.  WHEN exception_name THEN
```

```
58.          ...
59.  WHEN another_exception_name THEN
60.          ...
61.  WHEN OTHERS THEN
62.          ...
63.END;
```

```
/
```

64. The PL/SQL blocks can exist stand-alone (then they are immediately executed) or as bodies/definitions of stored procedures/functions or triggers (see below).
65. Each PL/SQL block will be compiled automaticaly and imediately after its definition (definition of each PL/SQL block should end by the line with single "/" characted). The compilation may result into errors and errorneous PL/SQL blocks cannot be execute (e.g., if an errorneous block defines a stored procedure or a trigger, the procedure or the trigger cannot be executed).
66. For more details, see Oracle Database PL/SQL Language Reference 11gR2. For examples, see Using Oracle PL/SQL.

Stored Procedures and Functions in Oracle Database

1. A stored procedure or a function is a named PL/SQL block similar to a procedure in other programming languages.

2. Each stored procedure or function has a header and a body. The header consists of the name of the procedure/function, the parameters or variables passed to the procedure/functions, and in the case of a function, also of a datatype of its return value. The body consists of a PL/SQL block (see above).
3. We can pass parameters to a procedure or a function in three ways:
    - IN parameters – to pass data into a PL/SQL block of the procedure/function only,
    - OUT parameters – to pass data out of a PL/SQL block of the procedure/function only,
    - IN OUT parameters – to pass data both into and out of a PL/SQL block of the procedure/function.
4. Declaration of a procedure have the following structure:

```
5. CREATE OR REPLACE PROCEDURE procedure_name(param1 [IN|OUT|IN OUT] datatype, ...)
6. IS
7.   -- declaration section (optional)
8. BEGIN
9.   -- execution section
10.  statements;
11.EXCEPTION
12.  -- exception section (optional)
END;
```

13.Declaration of a function have the following structure:

```
14.CREATE OR REPLACE FUNCTION function_name(param1 [IN|OUT|IN OUT] datatype, ...)
```

```
15.  RETURN datatype
16.IS
17.  -- declaration section (optional)
18.BEGIN
19.  -- execution section
20.  statements;
21.  RETURN return_variable;
22.EXCEPTION
23.  -- exception section (optional)
END;
```

24. Each stored procedure which has been compiled without errors can be executed in SQL by `EXECUTE procedure_name(value_for_param1, ...);` or in PL/SQL by `procedure_name(value_for_param1, ...);`
25. Each stored function which has been compiled without errors can be executed in SQL by `SELECT function_name(value_for_param1, ...) FROM dual;` (or in the similar way in common select/insert/update/delete statements) or in PL/SQL by `variable := function_name(value_for_param1, ...);`
26. For more details, see Oracle Database PL/SQL Language Reference 11gR2, CREATE PROCEDURE Statement and Oracle Database PL/SQL Language Reference 11gR2, CREATE FUNCTION Statement. For examples, see Using Oracle PL/SQL, Procedures.

Database Triggers in Oracle Database

1. A trigger is a PL/SQL block which is fired automatically when an associated type of insert, update, and/or delete statements is executed on an associated database table. The trigger can be fired for a statement just once or for each row of an associated table modified by the statements.
2. Declaration of a trigger have the following structure:

```
3. CREATE OR REPLACE TRIGGER trigger_name
4.   {BEFORE | AFTER | INSTEAD OF}
5.   {INSERT [OR] | UPDATE [OR] | DELETE}
6.   [OF col_name]
7.   ON table_name
8.   [REFERENCING OLD AS old_row_record_variable NEW AS new_row_record_variable]
9.   [FOR EACH ROW]
10.  [WHEN (condition)]
11.DECLARE
12.  -- declaration section (optional)
13.BEGIN
14.  ---- execution section
15.  statements;
16.  -- the statements can access values of an affected row before/after modification
17.  -- by a triggering statement in row variables :old_row_record_variable and :new_row_record_variable
18.  -- (or in variables :old and :new if the referencing section above is not declared)
19.EXCEPTION
20.  -- exception section (optional)
21.END;
```

22.In the declaration of a trigger, we can specify:

- if the trigger will be fired BEFORE, AFTER, or INSTEAD OF action performed by its associated types of statement,
- if the trigger will be associated with INSERT, UPDATE, and/or DELETE statements on its associated table,
- a database table (and optionaly also its particular column) associated with the trigger,
- row variables to access old and new values of each row affected by actions of the associated statements on the associated table; for UPDATE statements, :old can be used to access pre-update values while :new can be used to access post-update values; for INSERT statements, :new can be used to access inserted values (:old is NULL); and for DELETE statements, :old can be used to access deleted values (:new is NULL),
- if the trigger should be performed just once for triggering statements, or FOR EACH ROW affected by this statements,
- a particular condition when the trigger will be performed only,
- a PL/SQL code perfomed when the trigger is executed by the database. Moreover, BEFORE triggers can cancel the following action which would be performed by triggering statement

otherwise. This can be done by raising an exception, e.g. by `RAISE_APPLICATION_ERROR(ora_code, text_of_error_message);`

23."Before" and "after" triggers associated with a particular statement are fired in the following order:

- BEFORE statement trigger fires first (i.e., not "FOR EACH ROW" triggers),
- next, BEFORE row level trigger fires, once for each row affected (i.e., "FOR EACH ROW" triggers),
- then, AFTER row level trigger fires, once for each row affected (i.e., "FOR EACH ROW" triggers),
- finally, the AFTER statement level trigger fires (i.e., not "FOR EACH ROW" triggers).

24.Triggers are often used to check more complicated data constraints or to affect data manipulated by assciated statements. For example,

```
25.CREATE OR REPLACE TRIGGER author_id_autoincrement
26.  BEFORE INSERT ON author
27.  FOR EACH ROW
28.  WHEN (new.id is null)
29.BEGIN
30.  :new.id := author_seq.nextval;
31.END;
32./
33.
34.CREATE OR REPLACE TRIGGER book_isbn_check
35.  BEFORE INSERT OR UPDATE ON book
36.  FOR EACH ROW
```

```
37.DECLARE
38.  isbn VARCHAR2(10) := trim(replace(:new.isbn, '-'));
39.  isbn_numbers NUMBER := TO_NUMBER(substr(isbn, 1, 9));
40.  isbn_check_digit CHAR := TO_CHAR(substr(isbn, 10, 1));
41.  summe NUMBER := 0;
42.  i NUMBER;
43.BEGIN
44.  IF NOT LENGTH(isbn) = 10 THEN
45.          RAISE_APPLICATION_ERROR(-20001, 'The length of ISBN number supplied is invalid!');
46.  END IF;
47.  -- loop 9 times from 10 to 2 and sum individual numbers
48.  FOR i IN REVERSE 2..10 LOOP
49.          summe := summe + (i * TO_NUMBER(SUBSTR(isbn_numbers, 11 - i, 1)));
50.  END LOOP;
51.  -- check for 'X' as last digit and add its value to the sum
52.  IF isbn_check_digit = 'X' THEN
53.          summe := summe + 10;
54.  ELSE
55.          summe := summe + TO_NUMBER(isbn_check_digit);
56.  END IF;
57.  -- a factor of the sum should be 11
58.  IF NOT mod(summe, 11) = 0 THEN
59.          RAISE_APPLICATION_ERROR(-20002, 'The ISBN number supplied is invalid!');
60.  END IF;
61.  -- source:
62.  -- http://morph3us.org/blog/index.php?/archives/17-PLSQL-Trigger-for-validating-ISBN-numbers.html
63.END;
```

```
/
```

64. For more details, see Oracle Database PL/SQL Language Reference 11gR2, PL/SQL Triggers. For examples, see Using Oracle PL/SQL, Constraints and Triggers.

Indexing in Oracle Database

1. An index is an optional structure, associated with a table, that can sometimes speed data access (by reducing disk I/O). The database automatically maintains and uses indexes after they are created. The database also automatically reflects changes to data, such as adding, updating, and deleting rows, in all relevant indexes with no additional actions required by users. Retrieval performance of indexed data remains almost constant, even as rows are inserted.
2. A key is a set of table columns or expressions on which you can build an index. The such index will speed tests on values of the indexed keys, e.g., when they are used in WHERE caluse of SELECT queries. A composite index is an index on multiple columns in a table which should appear in the order that makes the most sense for the queries.
3. Indexes can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column or column, while nonunique indexes permit duplicates values in the indexed column or columns.
4. You can create an index by the following statements:

```
5. CREATE [UNIQUE] INDEX index_name1 ON table_name(column_name);
6. CREATE [UNIQUE] INDEX index_name2 ON table_name(column_name1, column_name2, ...);
```

```
CREATE [UNIQUE] INDEX index_name3 ON table_name(expression);
```

E.g., to speed searching authors by name in our example, we should defined the following indexes:

```
CREATE INDEX autor_family_name_idx ON author(family_name);
CREATE INDEX autor_given_names_idx ON author(given_names);
```

7. For more details, see Oracle Database Concepts 11gR2, Indexes and Index-Organized Tables and Oracle Database SQL Language Reference 11gR2, CREATE INDEX.
8. See also Oracle Database 12c PL/SQL.

Practice Exercise

1. Run the Oracle SQL Developer and connect to the Oracle database at BUT FIT.
2. Create the tables defined in this lecture and insert sample data into the tables (sample book, authors, and categories). Run the queries described in this lecture and evaluate the resulting data.
3. Feel free to experiment with creating another database objects and running queries, inesrts, updates, and deletes in the Oracle SQL Developer.