1. Preliminary information

Our civilization is a data civilization. Scientific theories are built on the basis of collected physical data, business data serves as the basis for making all decisions, and data is finally a value in itself - it is collected and searched. Due to the growing "resources" of data, a computer has become a natural tool for their collection and processing. Computer systems in which data is collected and managed are called Database Management Systems (DBMS). Among these systems, one of the most important is the system offered by Oracle.

1.1. Oracle database objects

The Oracle database is a collection of data stored in files. It has its logical structure (links between collected data) and physical structure (set of physical objects in which data is collected). The implementation of the Oracle database (instance) consists of a memory area called the System Global Area (SGA) and background processes that communicate with the SGA area and database files.

The basic structures (objects) stored in the Oracle database are tables. The following their types can be distinguished:

- Relational tables,
- Object-relational tables,
- Tables with an index structure,
- External tables,
- Partitioned tables
- Materialized Perspectives,
- Temporary tables,
- clusters
- Tables removed.

Relational tables contain data entered and processed by the user. Object-relational tables contain data about user-defined types for which the inheritance mechanism can be used. Tables with an index structure contain data stored in the index structure. External tables are used to access large external data without having to load it into the database. Partitioned tables divide their data into partitions that can be managed separately. Materialized perspectives contain the resulting data replica. In temporary tables, each user has the ability to "view" only the rows entered by him. One can save two tables in a cluster structure, to which queries are often directed together. Deleted tables allow you to quickly restore tables deleted using a command with special syntax.

In order to gain faster access to data stored in tables, Oracle uses structures (objects), stored in a database, called indexes. The following types of indexes can be distinguished:

- B*-tree indexes,
- Bitmap indexes,
- Indexes with a reverse key,
- Function indexes,
- Partitioned indexes,
- Text indexes.

B*-tree indexes are built based on binary trees. Bitmap indexes are used primarily during batch loading of data (e.g. in data warehouses). Reverse key indexes provide the ability to dynamically reverse indexed values before they are saved. Function indexes allow one to "base" an index on a function with attribute as an argument. Partitioned indexes are used to support partitioned tables. Text indexes are a collection of tables and indexes maintained by Oracle to enable advanced search within text.

In addition to relational tables and indexes, the Oracle system allows the use of many other structures such as views, procedures, functions, triggers, packages, snapshots, and users. Most of these structures will be described in this lecture.

The Oracle system has its own dictionary, i.e. a database in which the data about database (so-called metadata) are stored. These can be user data, their permissions, database object definitions, restrictions, etc.

Database objects that require physical space in permanent memory in Oracle obtain them as part of the so-called table space. The table space consists of one or more files. Each database object can be saved in one such file or it can be divided and saved in many files.

1.2. Data access methods

As part of the Oracle system, the following software methods of accessing data stored in the database can be distinguished:

- SQL language
- PL/SQL
- Dynamic SQL
- SQL*Plus,
- Java and JDBC,
- XML
- Object-oriented SQL and PL / SQL,
- Data Pump,
- SQL*Loader,
- External programs and procedures
- UTL_MAIL.

SQL is the primary query language for relational databases. The PL/SQL language enables creating procedures and functions (subprograms) stored in the database as well as triggers. As part of its syntax, one can use SQL commands. Dynamic SQL is used within PL/SQL and allows you to define commands that dynamically take into account the current state of the database or the current needs of the user. SQL*Plus is a simple interface that can support, among others, SQL commands and PL/SQL blocks. Thanks to the possibility of using Java and JDBC as part of the Oracle system, one can define subprograms stored in a database using this language. The Oracle system allows the use of XML types and interfaces. It also offers object extensions of SQL and PL/SQL with object-oriented capabilities (object types, object tables, methods). The Data Pump Import and Data Pump Export operations enable efficient data extraction and transfer to other databases. One can use SQL*Loader to quickly load files into Oracle tables. The Oracle database data can be accessed from external programs in which you can embed SQL commands. It is also possible to attach external subprogram libraries to Oracle. Used as part of PL/SQL, the UTL_MAIL package enables sending e-mail messages without the need for SMTP protocol.

1.3. Outline of the description of the reality used.

Most of the examples related to the lecture will be based on a database describing the fragment of reality presented below.

After many years of independence, cats of both sexes hunting in the village of Wólka Mała decided to organize themselves. So a herd was created, commanded by the greatest mouse hunter with the nickname Tiger. As part of the herd, under the leadership of the Tiger, an informal hierarchy of cats formed naturally - each cat knew which other cat it was subordinated to. The herd was additionally, administratively, divided into several, having a unique number and name, of the bands, each commanded by an outstanding mouse hunter nominated by the Tiger. Each band was given an independent area where the band's cats could organize their hunts. The Tiger and members of his band, due to their high position, had the privilege of hunting in the entire area controlled by the herd. For identification purposes, each cat was required to choose a unique nickname. The cat should also have a name. It was agreed that a member of the herd would be rewarded with a ration of mice every month for his contribution to maintaining the entire herd. This ration will be adequate to the function performed in the feline community. This function will define the lower and upper limit of mice ration.

Regardless of the size of the mice ration, the herd leader, for special merits, will be able to grant the cat, at his own discretion, additional mice ration. Cats hunted happily in their assigned areas, but from time to time there were incidents with representatives of other breeds. The participants of incidents, identified by their name, automatically became the personal enemies of the wronged cats and their degree of hostility and their species were carefully noted. As a warning to cats and infamy for enemies all these events were described (mandatory with their date). Assuming, however, that a real hunter can avoid known enemies, only the first cat incident with a specific enemy was recorded. Over time, cats noticed that handed to enemies bribes are able to reduce their alertness. For this reason, the sort of bribe, preferred by each enemy, was noted.

As a result of the analysis of the fragment of reality outlined above, a database schema was created consisting of five relations with the following schemas, presented in a predicate form:

Cats(<u>nickname</u>, name, gender, in_herd_since, mice_ration,
mice_extra, #function, #chief, #band_no)
Bands(<u>band_no</u>, name, site, #band_chief)
Functions(<u>function</u>, min_mice, max_mice)
Enemies(<u>enemy_name</u>, hostility_degree, species, bribe)
Incidents(#<u>nickname</u>, #<u>enemy_name</u>, incident_date, incident_desc)

where underline means primary key and # foreign key. The foreign key `function` in the `Cats` relation binds it with the `Functions` relation, the foreign key `band_no` bids it with the `Bands` relation and the foreign key `chief` binds it with herself (points to the superior of the cat). The foreign key `band_chief` in the `Bands` relation binds her with `Cats` relation. The foreign keys `nickname` and `enemy_name` in the `Incidents` relation bind her with the `Cats` and `Enemies` relations, respectively.

2. SQL language - Oracle dialect

SQL language (Structured Query Language) is recognized as specified by an international standard (SQL1 - ISO 1987, SQL2 - ISO 1992, SQL3 - ISO 1999, SQL 2003, SQL 2006, SQL 2008, SQL 2011) structural query language for relational databases. This standard is often extended in commercial implementations. Each such implementation of SQL is called its dialect. As part of this lecture, an SQL dialect proposed by Oracle will be presented. The database describing the reality of cats will be implemented in the Oracle company's DBMS and all examples will be made in the SQL Developer program offered by this company.

Generally, the SQL language consists of the following three components:

- DDL (Data Definition Language): a language for defining database objects with the basic CREATE, ALTER and DROP commands,
- DML (Data Manipulation Language): a language for manipulating data with basic INSERT, UPDATE, DELETE and SELECT commands and for controlling transactions with basic COMMIT and ROLLBACK commands,
- DCL (Data Control Language): a language for granting authority to objects and database operations with basic GRANT and REWOKE commands.

For the purposes of this lecture, a part of SQL containing the SELECT command has been separated from the DML component and named DQL (Data Query Language) component of SQL language. As part of the lecture SQL commands will be discussed in the order arising from the needs of the project accompanying the lecture, not in the systematic order shown in the above breakdown.

2.1. Relational tables

Relational tables are created, modified, and deleted using DDL commands of SQL language. The structure of the command that creates the table reflects the structure of the table specified in logical model of the database, and within the command syntax, restrictions on the table attributes and the entire structure of the created database (integrity bonds) are also defined. These restrictions include:

- mandatory attributes,
- unique attributes,
- domain bonds,
- entity integrity,
- reference integrity,
- propagation bonds,
- general bonds.

Depending on the SQL dialect, more or less of these constraints are defined directly using the DDL command that creates a relational table. In the Oracle database this command cannot define only general constraints.

The relational table (relation) in the Oracle database is created using the CREATE TABLE command. The most basic syntax for this command is as follows:

CREATE TABLE Relation_name
({attribute_name attribute_type [{**NOT NULL**} | <u>**NULL**</u>]
  [**DEFAULT** default_value]
  [{attribute_constraint [...]}] [, ...]}
  [, {relation_constraint [, ...]}]);

The relation consists of attributes with values limited by domain constraints. These constraints are defined by the attribute type and attribute constraints. NOT NULL defines the constraint associated with the mandatory attribute (by default the attribute is treated as optional). The DEFAULT clause specifies default value of the attribute. Attribute constraints (listed after spaces for attribute) apply to the implementation of the following restrictions: entity integrity (simple primary key), reference integrity (simple foreign keys) with propagation bonds, simple unique keys, and additional domain constraints. Relation constraints apply to constrains that cannot be defined within attribute constraints because they are based on more than one relation attribute. They relate to entity integrity (compound primary key), reference integrity (compound foreign keys) along with propagation bonds, compound unique keys, and other constraints based on at least two relation attributes.

The following basic types of relational table attributes can be distinguished in Oracle (alternative names of these types resulting from the SQL standard are also allowed):

Type name	Type description
`CHAR`	Fixed-length character strings (usually the default length is 255 characters)
`CHAR(w)`	Character strings with fixed length `w` (1≤`w`≤2000)
`VARCHAR2`	Character strings of variable length, however, not more than 4000 characters
`VARCHAR2(w)`	Character strings of variable length but not more than `w` characters (`w`≤4000)
`LONG`	Character data of variable length (up to 2 GB). Oracle recommends using `CLOB`
`CLOB`	Large character data (up to 128 TB, Oracle versions older than 10g to 4 GB)
`NUMBER`	Integers with a precision of 38 (precision is the number of significant digits)
`NUMBER(w)`	Integers with maximum precision `w` (`w`≤38)
`NUMBER(w,d)`	Real numbers with precision `w` and `d` decimal places (`w`≤38)
`DATE`	Date and time.
`RAW`	Binary data up to 2 KB in size
`LONG RAW`	Binary data up to 2 GB in size. Oracle recommends using `BLOB`.
`ROWID`	Represents the specific row address in the table

`BLOB`	Large volume binary data (up to 4 GB)
`BFILE`	Binary files. It allows to store read-only binary data in external files (outside the database)
`XMLTYPE`	Stores XML documents in `CLOB` columns (since Oracle 9i). Defined by the SYS module (`SYS.XMLTYPE`)
User data types	User-defined complex types (from Oracle 9i) using the base types specified above and other previously defined complex types

The attribute constraint and relation constraints have the following syntax:

[**CONSTRAINT** constraint_name] constraint

The following attribute and relationship comstraints are available:

Constraint syntax	Description of constraint
PRIMARY KEY [({attribute_name [, …]})]	For attribute constraint, it defines the attribute that acts as the primary key (entity integrity). For a compound key (relation constraint), a list of key attributes is specified. The constraint excludes with the UNIQUE constraint.
UNIQUE [({attribute_name [, …]})]	For attribute constraint, it defines an attribute that acts as a unique key (with NOT NULL constraint, it will be an alternative key). For a compound key (relation constraint), a list of key attributes is specified. The constraint excludes with the PRIMARY KEY constraint.

[**FOREIGN KEY** ({attribute_name [, …]})] **REFERENCES** relation_name [({attribute_name [, …]})]	Defines a foreign key (reference integrity). For the attribute constraint (simple foreign key), there is no FOREIGN KEY clause that lists the attributes that make up the key. Relation name means a name of the related relation followed by a list of attributes of this relation that act there as the primary, alternative, or unique key. For a simple key, this is a one-item list.
ON DELETE {**CASCADE** | **SET NULL**}	Constraint following the definition of a foreign key. It defines the binding propagation constraints (CASCADE - cascade deletion, SET NULL - deleting with NULL values inserting). If this constraint no occurs, limited deletion applies.
CHECK (condition)	The constraint concern domain constraints. It defines a condition that must be met by an attribute (attribute constraint) or several attributes (relation constraint) of the relation. In some SQL dialects condition can contain the subquery (Oracle does not do this).

No explicit name the restriction (CONSTRAINT clause) results the name SYS_Cn defined by the system, where n is the constraint number. Any attribute constraint can be defined as a relation constraint but not vice versa.

Descriptions of all user constraints can be found in the USER_CONSTRAINTS and USER_CONS_COLUMNS system views, while user relation descriptions in the USER_TABLES view.

The following examples of creating relational tables do not apply to the database that describes the reality of cats. This is due to the fact that the schema of this database will be implemented during the laboratory.

Task. *Define a relational table* `Persons` *with attributes:* `pesel`*,* `surname`*,* `first_name`*,* `gender` *with appropriate restrictions.*

```
CREATE TABLE Persons
(pesel NUMBER(11) CONSTRAINT pe_pk PRIMARY KEY
                  CONSTRAINT pe_pe_ch CHECK(pesel>10000000000),
 surname VARCHAR2(20) CONSTRAINT pe_sur_nn NOT NULL,
 first_name VARCHAR2(15) CONSTRAINT pe_fn_nn NOT NULL,
 gender CHAR(1) CONSTRAINT pe_ge_ch CHECK(gender IN ('W','M'))
);
```

Task. *Define a relational table* `Items_in_dok` *with attributes* `document_no`*,* `item_no`*,* `position_content` *with appropriate restrictions.*

```
CREATE TABLE Items_in_dok
(document_no VARCHAR2(20),
 item_no NUMBER(5),
 position_content LONG CONSTRAINT pos_con_nn NOT NULL,
 CONSTRAINT pos_con_pk PRIMARY KEY(document_no, item_no)
);
```

The primary key of the `Items_in_dok` relation is a table constraint (compound key) hence its definition is after definition of all attributes.

Task. *Let the data of honorary blood donors be collected in the* `Donors` *relation, data of donations collected from them in the* `Donations` *relation and the results of virological tests of donations in the* `Test_results` *relation. Define the listed relations (attributes, attribute constraints, and relevant relationships between them).*

```
CREATE TABLE Donors
(donor_no NUMBER(5) CONSTRAINT don_pk PRIMARY KEY,
 surname VARCHAR2(20) CONSTRAINT don_sur_nn NOT NULL,
 first_name VARCHAR2(15) CONSTRAINT don_fn_nn NOT NULL,
 gender CHAR(1) CONSTRAINT don_ge_nn NOT NULL
                CONSTRAINT don_pge_ch CHECK(gender IN ('W','M')),
 blood_group CHAR(2) CONSTRAINT don_blg_ch
                     CHECK (blood_group IN ('0','A','B','AB')),
 address VARCHAR2(50) CONSTRAINT don_add_nn NOT NULL
);
```

```
CREATE TABLE Donations
(donation_no VARCHAR2(20) CONSTRAINT dona_pk PRIMARY KEY,
 volume NUMBER(3) CONSTRAINT dona_vo_nn NOT NULL
                  CONSTRAINT dona_vo_ch CHECK (volume>0),
 collection_date DATE CONSTRAINT dona_colld_nn NOT NULL,
 donor_no NUMBER(5) CONSTRAINT dona_drno_nn NOT NULL
                    CONSTRAINT dona_drno_fk
                               REFERENCES Donors(donor_no)
);

CREATE TABLE Test_results
(test_id VARCHAR2(15) CONSTRAINT tr_pk PRIMARY KEY,
 result_wr CHAR(1) CONSTRAINT tr_rwr_ch
                   CHECK (result_wr IN ('-', '+','?')),
 result_hiv CHAR(1)  CONSTRAINT tr_rhiv_ch
                     CHECK (result_hiv IN ('-','+','?')),
 result_hbs CHAR(1)  CONSTRAINT tr_rhbs_ch
                     CHECK (result_hbs IN ('-','+','?')),
 result_ahcv CHAR(1) CONSTRAINT tr_rahcv_ch
                     CHECK (result_ahcv IN ('-','+','?')),
 test_data DATE,
 donation_no VARCHAR2(20) CONSTRAINT tr_dono_nn NOT NULL
                          CONSTRAINT tr_dono_fk
                             REFERENCES Donations(donation_no)
                             ON DELETE CASCADE
);
```

It is necessary to pay attention to the order in which relations are built by subsequent CREATE TABLE commands. First, relation are constructed that do not refer to any other relation, and only then relations dependent on them (through foreign keys). As previously mentioned, any column constraint can be defined as table constraint. For example, the equivalent definition of the Donations relation if all constraints would be defined as the table constraints, would has the following:

```
CREATE TABLE Donations
(donation_no VARCHAR2(20),
 volume NUMBER(3) CONSTRAINT dona_vo_nn NOT NULL,
 collection_date DATE CONSTRAINT dona_vo_nn NOT NULL,
 donor_no NUMBER(5) CONSTRAINT dona_vo_nn NOT NULL,
 CONSTRAINT dona_pk PRIMARY KEY (donation_no),
 CONSTRAINT dona_vo_ch CHECK (volume>0),
 CONSTRAINT dona_drno_fk FOREIGN KEY (donor_no)
                         REFERENCES Donors(donor_no)
 );
```

It should be noted here that after the PRIMARY KEY clause, the name of the attribute being the primary key is mentioned and the foreign key defined as a table constraint has an additional syntax element in the form of the FOREIGN KEY clause.

Relations of database defined by the CREATE TABLE command can be modified with the ALTER TABLE command. The need for such modification is usually associated with a change in the situation in the modeled reality. The most basic ALTER TABLE command syntax in Oracle SQL dialect is as follows:

```
ALTER TABLE Relation_name
 {ADD attribute_name attribute_type
   [{NOT NULL} | NULL]
   [DEFAULT default_value] attribute_constraint }
| {DROP COLUMN attribute_name
                    [CASCADE CONSTRAINTS}
| {ADD relation_constraint }
| {DROP {constraint_sort |
           CONSTRAINT constraint_name}}
| {DISABLE | ENABLE} { constraint_sort |
                         CONSTRAINT constraint_name }
| {MODIFY attribute_name attribute_type
                           [DEFAULT default_value]
                           [{NULL | NOT NULL}]}
```

With the above command one can:

- add a new attribute to the relation,
- remove attribute from relation. CASCADE CONSTRAINTS clause removes the attribute from all objects that refer to the removed attribute,
- add relation constraint,
- remove a constraint by its sort (available sorts are UNIQUE and PRIMARY KEY) or its name,
- temporarily disable or enable the constraint by its sort (available sorts are UNIQUE, PRIMARY KEY and ALL TRIGGERS) or name,
- modify the attribute specifying its default value and / or mandatory.

Modifying a relational table requires the following conditions:

- the attributes for which null values have been allowed cannot be modified,
- the relation cannot be extended with the new non-empty attribute,
- it is possible to reduce the size of the attribute type when the value of this attribute is empty in all relation rows.

Task. *Extend the* `Donor` *relation with a new attribute* `donor_status`*, specifying whether the donor is active (status 'A') or is a retired person (status 'R') or is already dead (status 'D').*

```
ALTER TABLE Donors ADD donor_status CHAR(1) DEFAULT 'A'
                      CONSTRAINT don_sta_ch
                      CHECK (donor_status IN ('A','R','D'));
```

Task. *For the* `volume` *attribute in the* `Donations` *relation, add a constraint that does not allow entering a donation larger than 999 ml.*

```
ALTER TABLE Donations
      ADD CONSTRAINT dona_vol_ch CHECK (volume<=999);
```

The constraint defined above is redundant (the attribute type volume specified by `NUMBER(3)` limits the volume of donation from above on the desired value), so it will be removed.

Task. *Remove from the* `Donations` *relation the constraint named dona_vol_ch for the* `volume` *attribute.*

```
ALTER TABLE Donations DROP CONSTRAINT dona_vol_ch;
```

The relation is removed from the database schema using the DROP TABLE command with the syntax:

DROP TABLE Relatio_name [**CASCADE CONSTRAINT**]

The optional CASCADE CONSTRAINT clause also deletes all objects that are dependent on the relation being removed.

2.2. SELECT query

The SELECT query is the only command of DQL part of SQL language. It allows the realization of operations known from the language of relation algebra, i.e. operation of selection, projection, cartesian product and joining. The basic syntax for the SELECT command on an Oracle system is as follows:

```
SELECT [DISTINCT | ALL] {expression [alias] [, ...]} | *
FROM {RelationViewName [alias]
        [join_operator  RelationViewName [alias]
         [ON joining_condition]] [, ...]}
[WHERE row_selection_condition]
[GROUP BY {expression [, ...]}
  [HAVING group_selection_condition]]
[ORDER BY {expression [DESC | ASC] [, ...]}]
```

Expressions in the above syntax are usually based on relation attributes (the attribute or constant is a special case of an expression). Alias means an alternative (placeholder) name. In the case of a relation or view alias, it replaces their name, which means that to their attributes one cannot referenced through this name. The SELECT clause of this command performs the projection operation. After the FROM clause, the relation/view from which the data are downloaded is determined. If several relation are listed after the comma, the data is taken from the cartesian product of these relations. If a join operator is used there, then the relation from which the data is downloaded is the result of the join realized with the join condition listed after ON clause. The WHERE clause implements the rows selection operation. The GROUP BY clause creates rows groups with the same values of the expression (s) listed after the clause. The HAVING clause performs the group selection operation. The ORDER BY clause that appears always last, sorts the resulting relation rows by the values of the expression (s) listed after the clause. Other elements of the syntax will be explained in the rest of the materials along with examples of queries.

The result of the SELECT query is always a relation, so this operation is closed. On Oracle, most clauses of this query may contain nested SELECT commands (so-called subqueries). This applies to the WHERE, HAVING, SELECT and FROM clauses. In addition, the nested SELECT command always occurs for vertical joins performed by the UNION, INTERSECT and MINUS operators.

The order in which the clauses are executed in the SELECT command is different from the one in the above syntax. Knowing this order may, in some cases, facilitate or even allow the construction of queries.

The SELECT clauses are executed in the following order:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

2.2.1. Simple queries

SELECT and FROM clause

The simplest form of the SELECT query contains only SELECT and FROM clauses. After the FROM clause, the relation/view from which the data is downloaded is defined (it can be the result of joining many relation/views), and after the SELECT clause, expressions are specified whose values should be displayed.

.

Task. *List the values of all attributes of the* `Functions` *relation.*

```
SELECT * FROM Functions;
```

```
FUNCTION   MIN_MICE              MAX_MICE
---------- --------------------- ----------------------
BOSS       90                    110
THUG       70                    90
CATCHING   60                    70
CATCHER    50                    60
CAT        40                    50
NICE       20                    30
DIVISIVE   45                    55
HONORARY   6                     25

 8 rows selected
```

The * sign in the above command means all attributes.

Task. *Specify the functions that cats perform in each band.*

```
SELECT DISTINCT band_no,function FROM Cats;

BAND_NO                FUNCTION
---------------------- ----------
1                      NICE
1                      DIVISIVE
3                      THUG
2                      CATCHING
1                      BOSS
2                      NICE
2                      CATCHER
4                      CATCHING
3                      CATCHING
2                      THUG
3                      CAT
3                      NICE
4                      CAT
4                      CATCHER

 14 rows selected
```

The above command performs the operation of projection of `Cats` relation to attributes `band_no` and `function`. The DISTINCT qualifier in the above query removes rows repetitions in the resulting relation. By default, the ALL (repetition) qualifier is assumed and does not need to be specified.

The simplest form of the SELECT command can be extended by:

- string, date or number constant (date or string in single quotes) called pseudo-column.

Task. *Specify the minimum and maximum mice ration connected with each function.*

```
SELECT function,'can eat from ',min_mice,' to ',max_mice
FROM Functions;

FUNCTION   'CANEATFROM'  MIN_MICE       'TO' MAX_MICE
---------- ------------- -------------- ---- --------
BOSS       can eat from  90              to  110
THUG       can eat from  70              to  90
CATCHING   can eat from  60              to  70
CATCHER    can eat from  50              to  60
CAT        can eat from  40              to  50
NICE       can eat from  20              to  30
DIVISIVE   can eat from  45              to  55
HONORARY   can eat from  6               to  25

 8 rows selected
```

- attribute/expression alias: this is the alternative name of the attribute or expression, mentioned in the SELECT clause after this attribute or expression, displayed as the name of the column and possibly used in the ORDER BY clause.

Task. *Specify the minimum and maximum mice ration connected with each function (use column aliases).*

```
SELECT function Role,'can eat from ' " ",
       min_mice "Min mice", ' to ' " ",max_mice "Max mice"
FROM Functions;

ROLE                     Min mice            Max mice
---------- ------------- -------------- ---- ---------
BOSS       can eat from  90              to  110
THUG       can eat from  70              to  90
CATCHING   can eat from  60              to  70
CATCHER    can eat from  50              to  60
CAT        can eat from  40              to  50
NICE       can eat from  20              to  30
DIVISIVE   can eat from  45              to  55
HONORARY   can eat from  6               to  25
 8 rows selected
```

- arithmetic expression.

***Task.** Determine the annual mice consumption for each cat.*

```
SELECT name,(NVL(mice_ration,0)+NVL(mice_extra,0))*12
            "Eats annually"
FROM Cats;

NAME             Eats annually
---------------- ----------------------
BARI             672
MICKA            864
LUCEK            516
SONIA            660
LATKA            480
DUDEK            480
MRUCZEK          1632
CHYTRY           600
KOREK            1056
BOLEK            1116
ZUZIA            780
RUDA             768
PUCEK            780
PUNIA            732
BELA             624
KSAWERY          612
JACEK            804
MELA             612

 18 rows selected
```

The NVL function returns the value of the first argument if this argument is different from NULL, otherwise it returns the value of the second argument.

- concatenation operator: operator with the symbol ||, which joins two values of expressions into single string of characters.

***Task.** Specify the minimum and maximum mice ration connected with each function.*

```
SELECT function||' can eat from '||min_mice||' to '||max_mice||
       ' mice per month' "Function possibilities"
FROM Functions;
```

```
Function possibilities
--------------------------------------------------------------
BOSS can eat from 90 to 110 mice per month
THUG can eat from 70 to 90 mice per month
CATCHING can eat from 60 to 70 mice per month
CATCHER can eat from 50 to 60 mice per month
CAT can eat from 40 to 50 mice per month
NICE can eat from 20 to 30 mice per month
DIVISIVE can eat from 45 to 55 mice per month
HONORARY can eat from 6 to 25 mice per month
 8 rows selected
```

WHERE clause

The WHERE clause enables rows selection operations on the relation specified by the FROM clause. This selection takes place according to the condition (value of the logical expression) placed after the WHERE clause.

Task. *Find the names of all cats who perform the function NICE.*

```
SELECT name
FROM Cats
WHERE function='NICE';
```

```
NAME
---------------
MICKA
SONIA
RUDA
BELA
```

When constructing a selection condition, the following operators can be used:

=, !=, >, >=, <, <=, IS NULL, BETWEEN ... AND ..., IN (set), LIKE, NOT, AND, OR

The operators listed above, within each SQL expression, are executed according to the following priority (from highest to lowest):

1. =, !=, <, >,<=, >=, BETWEEN ... AND ... , IN, LIKE, IS NULL
2. NOT
3. AND
4. OR

In systems implementing the relational model, and thus in the Oracle system, an additional special value incompatible with this model is introduced, representing the lack of information about the value of the attribute. It is a NULL value (in fact it is a lack of value!), different from zero for numeric types or, e.g., from an empty sign for character types (in Oracle this value supported by NVL function). It introduces, de facto, trivalent logic (TRUE, FALSE, NULL) instead of traditional divalent logic (TRUE, FALSE). Therefore, the following rules apply to the computing of the values of the logical expressions with NULL arguments:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NULL values may have attributes for which their optionality is allowed. In this situation, the value of the arithmetic expression with the argument of NULL value is NULL. An expression with a relational operator (e.g., <) that contains an arithmetic expression of NULL value also has a NULL value. The NULL value in arithmetic expressions therefore requires special handling (e.g. via the NVL function).

Task. *Find cats that don't get extra mice.*

```
SELECT nickname,gender||'    ' "GENDER"
FROM Cats
WHERE mice_extra IS NULL;
```

```
NICKNAME        GENDER
--------------- ------
TUBE            M
ZERO            M
EAR             W
SMALL           M
BOLEK           M
FAST            W
REEF            M
HEN             W
MAN             M
CAKE            M
LADY            W

 11 rows selected
```

Task. *Find cats with ration of mice between 50 and 60.*

```
SELECT nickname
FROM Cats
WHERE mice_ration BETWEEN 50 AND 60;

NICKNAME
---------------
TUBE
BOLEK
MAN
LADY
```

Task. *Find the names of cats performing the function of THUG or CATCHING, whose direct boss is TIGER.*

```
SELECT name
FROM Cats
WHERE function IN ('THUG','CATCHING') AND chief='TIGER';

NAME
---------------
KOREK
BOLEK
PUCEK
```

Task. *Find cats whose have as a second letter O in their name.*

```
SELECT name
FROM Cats
WHERE name LIKE '_O%';
```

```
NAME
---------------
SONIA
KOREK
BOLEK
```

The '_' character in the pattern means any character and the '%' character means any rest of the string. If it is necessary to check the presence of '_' or '%' in the string, they should be placed after the citation character defined in the ESCAPE clause, e.g .:

```
WHERE name LIKE '_&_U_&%U%' ESCAPE '&'
```

The above entry means that the first character of the name is arbitrary, the second character is _, the third character is U, the fourth character is any, the fifth character is %, the sixth character is U and the remaining characters are any.

Task. *Specify the nickname, function, ration of mice and ration extra for cats with not null ration extra of mice, whose ration of mice exceeds 70 or that have the function NICE.*

```
SELECT function,NVL(mice_ration,0) "MICE",mice_extra
FROM Cats
WHERE mice_extra IS NOT NULL
      AND
     (NVL(mice_ration,0)>70 OR function='NICE');

FUNCTION   MICE                   MICE_EXTRA
---------- ---------------------- ----------------------
NICE       25                     47
NICE       20                     35
BOSS       103                    33
THUG       75                     13
THUG       72                     21
NICE       22                     42
NICE       24                     28

 7 rows selected
```

ORBER BY clause

The rows of resulting relations of the SELECT query are usually unordered. The ORDER BY clause is used to explicitly order them against the value of the attribute/expression (list of attributes and/or expressions). The following elements may appear in the clause: attribute identifier, expression, alias of the expression or of attribute from the SELECT clause, number of the expression/attribute in the SELECT clause. The default ordering direction is ascending (ASC). The descending direction is defined in ORDER BY clause by the word DESC after attribute/expression, by which ordering is performed. Ordering can be also carried out implicitly. It is part of the following operations: CREATE INDEX, DISTINCT, GROUP BY, ORDER BY, INTERSECT, MINUS, UNION, joining of unindexed relations.

Task. *Display enemy data according decreasing hostility.*

```
SELECT hostility_degree "How dangerous",
       enemy_name "Enemy name"
FROM Enemies
ORDER BY hostility_degree DESC;

How dangerous          Enemy name
---------------------- ---------------
10                     KAZIO
10                     WILD BILL
7                      UNRULY DYZIO
5                      SLYBOOTS
4                      DUN
3                      BASIL
2                      REKS
1                      BETHOVEN
1                      SLIM
1                      STUPID SOPHIA

 10 rows selected
```

***Task**. Display data of cats for which the ration of mice exceeds 60. Sort data first ascending by gender and name of the band and then descending by date of join to the herd and then ascending by function name.*

```
SELECT nickname "Nickname",gender "Gender",
       band_no "Band",in_herd_since "Join date",
       mice_ration "Eats"
FROM Cats
WHERE mice_ration>60
ORDER BY 2,"Band",in_herd_since DESC,function;

Nickname        Gender Band       Join date              Eats
--------------- ------ ---------- ---------------------- ----
TIGER           M      1          2002-01-01             103
CAKE            M      2          2008-12-01             67
BALD            M      2          2006-08-15             72
ZOMBIES         M      3          2004-03-16             75
REEF            M      4          2006-10-15             65
FAST            W      2          2006-07-21             65
HEN             W      3          2008-01-01             61

 7 rows selected
```

Pay attention that ordering is a time-consuming operation and should not be abused.

2.2.2. Grouping queries

Information about not a single row but a whole row group is often interesting for the user. To obtain this information, grouping should be performed.

GROUP BY clause

The GROUP BY clause allows grouping of relation rows due to the value of the attribute/expression (list of attributes and/or expressions) listed after the clause (the each group includes rows with the same attribute/expression value) and allows to display one result row, representing each group. The SELECT clause, if grouping is used, can only contain: attributes/expressions listed after the GROUP BY clause, aggregate functions, pseudo-columns, or expressions, which include these elements. If grouping is performed by of an attribute/expression that can have a null value (NULL), this value is treated as an additional value for the attribute/expression (an associated group is created).

Task. *Find nicknames of cats with subordinates.*

```
SELECT chief
FROM Cats
GROUP BY chief;

CHIEF
---------------

TIGER
ZOMBIES
BALD
REEF
HEN

 6 rows selected
```

The sixth row selected is for the group in which the `chief` attribute has no value (NULL).

Task. *Find the number of female cats performing a specific function in each band.*

```
SELECT COUNT(*)||
       ' number of female cats in the '||band_no||
       ' band with the function of '||function
       "Statistics for functions"
FROM Cats
WHERE gender='W'
GROUP BY band_no,function;

Statistics for functions
-------------------------------------------------------------------
2 number of female cats in the 1 band with the function of NICE
1 number of female cats in the 2 band with the function of CATCHING
1 number of female cats in the 2 band with the function of NICE
1 number of female cats in the 3 band with the function of CATCHING
1 number of female cats in the 3 band with the function of NICE
1 number of female cats in the 4 band with the function of CAT
1 number of female cats in the 4 band with the function of CATCHER

 7 rows selected
```

To find the number of female cats, the aggregate function COUNT was used to determine the number of rows in a group. This function has the syntax:

COUNT (* | {[**DISTINCT** | **<u>ALL</u>**] expression})

The COUNT function always counts rows in a group. If the argument of the function is *, all rows are counted, if the argument is an expression (its special case is e.g. an attribute), only those rows for which the expression is different from NULL are counted (repetitions of the value of the expression are also taken into account - default ALL), if additionally the DISTINCT qualifier appears before the expression, only rows for which the expression is different from NULL are counted, however, rows with a repetitive value of the expression are omitted in the count. In addition to the COUNT functions, the following aggregate functions are typically implemented in database systems:

SUM ([**DISTINCT** | **<u>ALL</u>**] expression) - returns the sum of the values of non-NULL expressions taken from each row of the group; DISTINCT omits the repetitive expression value from the calculation (the default is ALL),
AVG ([**DISTINCT** | **<u>ALL</u>**] expression) - returns the arithmetic average of the values of non-NULL expressions taken from each row of the group; DISTINCT omits the repetitive expression value from the calculation (the default is ALL),
MAX (expression) - returns the maximum value among non-NULL expression values, taken from each row of the group,
MIN (expression) - returns the minimum value of non-NULL expressions values taken from each row of the group.

The SQL dialect implemented in Oracle supplements this set of aggregate functions with many other functions, most often of a statistical nature. Aggregation functions can generally only be used in the SELECT and HAVING clauses of a SELECT query. No grouping is required to use them in the SELECT clause. Aggregation functions then work on the entire result relation and only they (or pseudo-columns) can occur in this clause.

Task. *Find the average consumption of mice for each gender (including additional rations).*

```
SELECT DECODE(gender,'W','Female cats','Male cats')"Gender",
       AVG(NVL(mice_ration,0)+NVL(mice_extra,0))
       "Average consumption"
FROM Cats
GROUP BY gender;

Gender      Average consumption
----------- ----------------------
Female cats 57,5
Male cats   68,9
```

Used in the above code, characteristic of Oracle, the DECODE function, realizes the CASE conditional statement. The value of the first argument of the function is compared to the value of the first argument of the subsequent pairs of arguments. If equality occurs, the value of the second argument of the corresponding pair is returned, if not then the next pair is checked. In case of no equality for any pair, the value of the last argument of the function is returned, if it occurs (if it does not occur, NULL is returned). The DECODE function can be nested freely. In the above code, the value of the `gender` attribute is compared to the character constant `'W'` (first element of the first pair). If equality occurs, the string `'Female cats'` (the second element of the first pair) is returned, otherwise the string constant `'Male cats'` (the last argument of the function) is returned due to the lack of subsequent pairs. Instead of the DECODE function, you the CASE function can be use, according to the ANSI SQL standard. Below is a solution to the above task using this function.

```
SELECT CASE gender
         WHEN 'W' THEN 'Female cats'
         ELSE 'Male cats'
       END "Gender",
       AVG(NVL(mice_ration,0)+NVL(mice_extra,0))
       "Average consumption"
FROM Cats
GROUP BY gender;
```

The value of the expression after CASE corresponds to the first argument of the DECODE function, the values after WHEN and THEN define subsequent pairs (the number of WHEN ... THEN elements can be any), the value after ELSE corresponds to the last argument of the DECODE function (ELSE element may not appear).

HAVING clause

The HAVING clause is used to select groups resulting from the grouping operation (GROUP BY) and cannot occur without the GROUP BY clause. The condition after HAVING determines which groups are to be selected. This condition can be built only on the basis of the attribute/expression (attributes/expressions) appearing in the GROUP BY clause, constants or/and on the basis of aggregate functions.

Task. *Find bands with "mice chimneys" (the ration of mice of a small number of cats far exceeds the ration of other cats).*

```
SELECT band_no "Chimney band",
       AVG(mice_ration) "Average ration",
       (MAX(NVL(mice_ration,0))+
       MIN(NVL(mice_ration,0)))/2
       "(MAX+MIN)/2"
FROM Cats
GROUP BY band_no
HAVING (MAX(NVL(mice_ration,0))+
        MIN(NVL(mice_ration,0)))/2>
        AVG(NVL(mice_ration,0));
```

```
Chimney band           Average ration       (MAX+MIN)/2
---------------------- -------------------- ------------------
1                      50                   62,5
4                      49,4                 52,5
```

CONNECT BY and START WITH clauses

The basic SELECT command syntax presented earlier is supplemented in the Oracle system with additional CONNECT BY and START WITH clauses. They are most often used when, within the conceptual model of the database, there exists an entity (class) in relationship with itself (hierarchical relationship), which relationship defines the hierarchy, e.g. superior - subordinate. The implementation of such a relationship in the logical model of the database is a foreign key associated with the primary key of the same relation. The Oracle system uses such a relationship to build, using these clauses, a row tree that reflects this hierarchy. The START WITH clause specifies the condition indicating the row which is to be the root of the tree (if n rows satisfies the condition, n trees are built) and the CONNECT BY clause the condition defining the way the tree is built, i.e. determining which row should be attached as the current leaf of node. Within the condition, the CONNECT BY clause is followed by the PRIOR keyword indicating the so-called the parent attribute (in the row of the current node) from which the value is taken to compare with the value of the second attribute of condition, called the child attribute (from the row that can become a leaf). The tree is built by selecting, as a leaves, the rows that meet the condition after CONNECT BY. Most often, this condition is based on the equality of a foreign key, representing a hierarchical relationship, with the primary key of the same relation. As part of a tree-building query, one can use:

– level pseudo attribute - defines the "depth" of the operated row in the tree,
– operator CONNECT_BY_ROOT attribute - returns, for the indicated attribute from the currently supported row in the tree, the value of this attribute in the row of the root,
– the SYS_CONNECT_BY_PATH function (attribute, separator) - returns, for the supported row, the path in the tree from the row of the root to this row, in the form of a string built from the successive values of the indicated attribute at each intermediate node, each value separated by a string separator.

***Task.** Determine the hierarchy in a herd of cats from the herd leader. In the built tree, skip the cat named KOREK together with all his subordinates and cats with the function NICE.*

```
SELECT name "Name",level "Position",
       band_no "Band",NVL(mice_ration,0) "Eats"
FROM Cats WHERE function!='NICE'
CONNECT BY PRIOR nickname=chief AND name!='KOREK'
START WITH chief IS NULL
ORDER BY band_no,level;

Name            Position      Band          Eats
--------------- ------------- ------------- -------------
MRUCZEK         1             1             103
CHYTRY          2             1             50
BOLEK           2             2             72
JACEK           3             2             67
ZUZIA           3             2             65
BARI            3             2             56
PUCEK           2             4             65
LATKA           3             4             40
MELA            3             4             51
KSAWERY         3             4             51
DUDEK           3             4             40

 11 rows selected
```

***Task.** For each cat belonging to the subtrees with roots ZOMBI and RAFA (nicknames of cats) present the nickname of the cat from the root of the subtree and, in the form of further nicknames, paths from the nickname of the cat from the root to the nickname of the served cat.*

```
SELECT nickname "Cat",
       DECODE(CONNECT_BY_ROOT nickname,
              nickname,NULL,CONNECT_BY_ROOT nickname) "Chief",
       SYS_CONNECT_BY_PATH(nickname,'/') "Nickname path"
FROM Cats
CONNECT BY PRIOR nickname=chief
START WITH nickname IN ('ZOMBIES','REEF');
```

```
Cat             Chief           Nickname path
--------------- --------------- ------------------------------
REEF                            /REEF
EAR             REEF            /REEF/EAR
LADY            REEF            /REEF/LADY
MAN             REEF            /REEF/MAN
SMALL           REEF            /REEF/SMALL
ZOMBIES                         /ZOMBIES
FLUFFY          ZOMBIES         /ZOMBIES/FLUFFY
HEN             ZOMBIES         /ZOMBIES/HEN
ZERO            ZOMBIES         /ZOMBIES/HEN/ZERO

 9 rows selected
```