1. SELECT: BASIC

2. SELECT: CONDITIONS

3. SINGLE-LINE FUNCTIONS

arg={arrow or literal but one datatype}

- `NVL(arg, arg_instead_of_NULL)`
- `NVL2(arg, arg_instead_of_NOT_NULL, arg_instead_of_NULL)`
- `NULLIF(arg1, arg2)`
  - if (arg1==arg2) then NULL, else arg1
- `COALESCE(arg1, ..., argN)`
  - if (arg1!=NULL) then arg1, if (arg1==NULL and arg2!=NULL) then arg 2,... else argN

arg={str|arrow}

- `LOWER(arg)`
- `UPPER(arg)`
- `INITCAP(arg)`
- `CONCAT(arg1, arg2)`
- `SUBSTR(arg, entry_num, num_of_sym)`
- `LENGTH(arg)`
- `INSTR(arg, str_for_search, [search_start_pos], [appearance_num])`
- `LPAD(arg, num_of_sym, [str])`
  - `''` by default, num_of_sym is a num of symbols with arg inclusive
- `RPAD(arg, num_of_sym, [str])`
- `TRIM([[LEADING | TRAILING | BOTH] 'sym' FROM] arg)`
  - (BOTH `''` FROM arg) by default
- `REPLACE(arg, from_str, [to_str])`
  - to_string==NULL by default (delete symbols)

- `TO_DATE(arg, [data_format])`
- `TO_NUMBER(arg, [num_format])`

arg={num|arrow}

- `ROUND(arg, num_after_dot)`
- `TRUNC(arg, num_after_dot)`
- `MOD(divisible_arg, division_arg)`
- `TO_CHAR(arg, num_format)`

arg={data|arrow|SYSDATE}

- `MONTHS_BETWEEN(arg1, arg2)`
- `ADD_MONTHS(arg, num_of_months)`
- `NEXT_DAY(arg, 'D')`
  - next 'D' after arg
- `LAST_DAY(arg)`
  - of current month
- `ROUND(arg, {round_date_par})`
- `TRUNC(arg, {round_date_par})`
- `TO_CHAR(arg, data_format, ['NLS_DATE_LANGUAGE = english'])`

conditional:

- `CASE [arrow] WHEN condition THEN expression [WHEN cond2 THEN exp2 ... ELSE else_exp] END`
- `DECODE(exp, exp1_to_compare, output1, [..., else_output])`
  - outputN if exp==expN

4. MULTI-LINE FUNCTIONS

arg = {arrow}

- `AVG([DISTINCT|ALL] arg)`
  - ALL by default

- `COUNT([*|DISTINCT|ALL] arg)`
  - * mean all str even NULL
- `MAX([DISTINCT|ALL] arg)`
- `MIN([DISTINCT|ALL] arg)`
- `STDDEV([DISTINCT|ALL] arg)`
  - standard deviation
- `SUM([DISTINCT|ALL] arg)`
- `VARIANCE([DISTINCT|ALL] arg)`
  - dispersion

5. JOINS

Common

- u can use tbl_alias like "FROM table t" (t is new name of table)

Standart syntaxis

- t = table

```
SELECT ...
FROM
  t1 NATURAL JOIN t2;
```

- join by all same arrows

```
SELECT ...
FROM
  t1 JOIN t2 USING (arrow);
```

- if several same arrows, but we don't need all

```
SELECT ...
FROM
  t1 {LEFT|RIGHT|FULL} {INNER|OUTER} JOIN t2 [ON (condition)];
```

- inner by default

- if different names of arrows (t1.arrow1 = t2.arrow2)
- u can do several joins one after the other
- cartesian product if there is not condition

```
SELECT ...
FROM
  t1 CROSS JOIN t2;
```

- cartesian product

Oracle syntaxis

```
SELECT ...
FROM
  t1, t2 WHERE t1.arrow1 = t2.arrow2;
```

```
SELECT ...
FROM
  t1, t2 WHERE t1.arrow BETWEEN t2.arrow1 AND t2.arrow2;
```

```
SELECT ...
FROM
  t1, t2 WHERE t1.arrow1 (+) = t2.arrow2;
```

- right outer join

```
SELECT ...
FROM
  t1, t2 WHERE t1.arrow1 = t2.arrow2 (+);
```

- left outer join

6. SUBQUERIES

7. SET OPERATORS

Union

```
SELECT... UNION [ALL] SELECT...;
```

Intersect

```
SELECT... INTERSECT SELECT...;
```

Minus

```
SELECT... MINUS SELECT...;
```

8. DML

Insert

```
INSERT INTO table [(column1, column2, ..., columnN)]
VALUES (val1, val2, ..., valN);
```

- if not all columns then other arr insert NULL
- subselect can be instead of table
- default order
- if there's not schema then NULL explicitly
- val can be subselect
- if subselect then can be several rows

Update

```
UPDATE table
SET column1 = val1, column2 = val2, ..., columnN = valN
[WHERE conditions];
```

- subselect can be instead of table
- if no WHERE then all rows will be updated
- vals can be subselects

Delete

```
DELETE [FROM] table
[WHERE conditions];
```

- subselect can be instead of table
- if no WHERE then all rows will be deleted

DDL: Truncate

```
TRUNCATE TABLE name;
```

- delete all rows

TCL

- include DML and DCL

```
SAVEPOINT name;
```

```
ROLLBACK [TO SAVEPOINT name];
```

- auto when emergency-shutdown

```
COMMIT;
```

- auto when DDL/DCL/normal-shutdown

9. DDL

CREATE TABLE

```
CREATE TABLE [scheme.]name
    (column1 DATATYPE [DEFAULT expression]
        [CONSTRAINT ...],
    ...
    [columnN ...],
    [CONSTRAINT constr1 ...], ...
    [CONSTRAINT constrN ...] );


CREATE TABLE [scheme.]name
    [(column1, ...)]
    AS subSELECT;
```

- it's necessary to have privelegy CREATE TABLE (only in own scheme) or CREATE ANY TABLE (in any scheme)
- when using subSELECT then columns in staples must match with subSELECT (and you can't specify datatype)

Tablename rools:

- first symbol is letter
- 1-30 bytes (e.g. encoding of 2 bytes = max 15 symbols)
- A-Z a-z 0-9 _ $ # (maybe different national alphabets)
- different names (for 1 user)
- **not** reserved by oracle names
- can be enclosed in " " (better not)
- `scheme.name` if table relate to another user
- `DEFAULT expression` may be literal/expression/function/SYSDATE (same datatype) and **NOT** another column/pseudocolumn (ROWNUM, ROWID, NEXTVAL, CURVAL)

DATATYPE:

Character:

- `VARCHAR2(size)`

    - dedicated memory $\leq$ size (= size of literal)
    - best memory usage
    - 1 sym = 1 byte
- `CHAR(size)`
    - dedicated memory = size
    - best perfomance
    - 1 sym = 1 byte
- `NVARCHAR2(size)` and `NCHAR(size)`
    - same but 1 sym = 2 byte
- `LONG`
    - 2 GB
    - outdated datatype
    - only 1 per table
- `CLOB` (character large object)
    - instead of LONG
    - max 32 GB
    - several per table

Number:

- `NUMBER(p,s)`
    - p - number of digit (max 38)
    - s - digits after dot (can be < 0)

Binary:

- `RAW` and `LONG RAW`
    - outdated datatype
- `BLOB`
    - instead of LONG RAW
    - 32 GB
    - store in database files
- `BFILE`
    - 32 GB
    - external file (OS)
    - realised like reference to file

Datetime:

- `DATE`
    - 01.01.4713 BC - 31.12.9999

    - precision: sec
- `TIMESTAMP[(precision_in_sec_frac=3)] [WITH [LOCAL] TIME ZONE]`
    - precision: fractions of sec (ns)
- `INTERVAL YEAR [(prec_in_years=2)] TO MONTH`
- `INTERVAL DAY [(prec_in_days=2)] TO SECOND [(prec_in_sec_frac)]`

Other:

- `ROWID, JSON, XML`

Constraints

Types:

- `NOT NULL`
- `UNIQUE`
- `PRIMARY KEY`
- `FOREIGN KEY ... REFERENCES table[(column)] [ON DELETE {CASCADE|SET NULL}]`
- `CHECK (condition)`

Other:

- if you don't assign constraint name, it will be assigned by default in format "SYS_Cn"
- column-level constraint:

```
CREATE TABLE ...
(column ... [CONSTRAINT constraint_name] CONSTRAINT_TYPE, ...);
```

- table-level constraint:

```
CREATE TABLE ...
(column ..., [CONSTRAINT constraint_name] CONSTRAINT_TYPE (columns));
```

- constraint that applies to several columns only in **table-level**
- NOT NULL only in **column-level**
- UNIQUE allows NULL
- FK refer to any candidate key
- if FK refer to PK then it's not necessary to specify column
- column-level FOREIGN KEY:

```
CREATE TABLE ...
```

```
(column ... [CONSTRAINT constraint_name] REFERENCES table[(column)], ...);
```

- table-level FOREIGN KEY:

```
CREATE TABLE ...
(column ..., [CONSTRAINT constraint_name] FOREIGN KEY (columns) REFERENCES table[(columns)]);
```

- ON DELETE CASCADE = delete child with parent
- ON DELETE SET NULL = set child NULL with delete parent
- CHECK doesn't allow:
  - function condition (SYSDATE, UID, USER, USERENV)
  - pseudocolumn (CURRVAL, NEXTVAL, LEVEL, ROWNUM, ROWID)
  - subSELECTs

OTHER DDL

```
ALTER TABLE name;
```

- if PURGE then table will be permanently deleted and not to recyclebin

```
DROP TABLE name [PURGE];
```

10. OBJECTS

Objects types

- table
- view
- sequence
- index
- synonym

View

- simple (1 table, no functions, no data groups, support DML)

- complex (1+ table, functions, data groups, support DML not always)
- rows cannot be deleted/updated/inserted:
    - multi_line_functions,
    - GROUP BY
    - DISTINCT
    - ROWNUM
- rows cannot be updated/inserted:
    - expression-columns
- rows cannot be inserted:
    - if no NOT NULL columns (because if PK not in view then insert row will contain NULL)
- FORCE if table doesn't exist (NOFORCE by default)
- WITH CHECK OPTION: DML won't work if condition in subSELECT of view will be violated
- WITH READ ONLY: DML is prohibited

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW name
  [(alias1, ... [aliasN])]
    AS subSELECT
  [ WITH CHECK OPTION [CONSTRAINT name] ]
    [ WITH READ ONLY [CONSTRAINT name] ];


DROP VIEW name;


DESC[RIBE] view;


SELECT ...
FROM view
...;
```

Sequence

- generate unique numbers automiticly (often for PK)
- shared usage (if users have privileges)
- it can use cache memory (to speed up access)
- skipping values reasons:
    - rollback transaction
    - system error

- using in different tables simultaneously

```
CREATE SEQUENCE name
    [INCREMENT BY n=1]
    [START WITH n=1]
    [{MAXVALUE n | NOMAXVALUE}]  --NOMAXVALUE by default
    [{MINVALUE n | NOMINVALUE}]  --NOMINVALUE by default
    [{CACHE n=20 | NOCACHE}]
    [{CYCLE | NOCYCLE}];  --NOCYCLE by default (not use CYCLE for PK)
```

- any order of commands
- `NEXTVAL` -- next free number in seq (unique at each request)
- `CURVAL` --current val of seq (it can has val only after using NEXTVAL)
- `seq.NEXTVAL / seq.CURVAL`

```
ALTER SEQUENCE name
    [...];
```

- only for owner of sequence (or owner of alter privilegy)
- generated nums can not be altered (and initial val)

```
DROP SEQUENCE name;
```

Index

- for speed up string selection with a pointer
- independent on table
- oracle create and support automatically (PK, UNIQUE)
- handle (user's non-unique index)
- most popular indexes:
  - B-tree (big num)
  - Bitmap (small num)
- good reasons:
  - big range of values
  - many NULLS
  - several columns often together in WHERE conditions
  - big table, a little % of data in SELECT
- bad reasons:

    - rare using of columns
    - little table, big % of data in SELECT
    - table updates often
    - columns are arguments of function / parts of expressions
- if function is implemented over column then index won't work (need new index over column function)

```
CREATE INDEX name
ON table (arr1 [, arrN]);


DROP INDEX name;
```

Synonym

- table of another users without username
- change long names
- PUBLIC = common access

```
CREATE [PUBLIC] SYNONYM name
FOR obj;


DROP SYNONYM name;
```

11. DICTIONARY

Common

- metadata about user data
- dictionary contains of base table (not used) and views of dictionary (`PREFIX_NAME`)
- dictionary belong to SYS user

Prefix

- `USER` user data
- `ALL` data that user have privileges

    - column "owner" was added
- `DBA` all data (for admin)
- `V$` efficiency data, dynamic views
    - data not only from tables but also from RAM
    - database can be not fully open to use this views

Views

- `dictionary` tables and views of dictionary
- `user_objects` user scheme
    - object_name
    - object_type
    - created
    - last change date
    - status (valid/invalid)
- `all_objects`
- `user_tables` about user tables
    - table_name
    - tablespace_name
    - cluster_name
        - if several tables was often used in one SELECT then one segment is allocated to several tables (cluster)
    - iot_name
        - index organized table (data is stored with indexes)
- `user_tab_columns` about user columns
    - table_name
    - column_name
    - data_type
    - data_type_mod
    - data_type_owner
    - data_length
    - data_precision
    - data_scale
    - nullable
    - column_id
    - default_length

    - data_default
- `user_constraints` common info about constaints, NOT about columns
    - owner
    - constraint_name
    - constraint_type
        - C (check, included not null)
        - U (unique)
        - P (pk)
        - R (references (fk))
        - V (check option)
        - O (read only)
    - table_name
    - search_condition (check condition)
    - r_owner (user of FK)
    - r_constraint_name (name of FK)
    - delete_rule (cascade/set null)
    - status
- `user_cons_columns` about constraints' columns
    - owner
    - constraint_name
    - table_name
    - column_name
    - position (if several columns in constraint then every column has position)
- `user_views`
    - view_name
    - text_length
    - text
- `user_sequences`
    - sequence_name
    - min_value
    - max_value
    - increment_by
    - cycle_flag
    - order flag
    - cache_size
    - last_number (last free number if nocache or last used number if cache)
- `user_synonyms`

    - synonim_name
    - table_owner
    - table_name
    - db_link (object that allows join several dbs)
- `user_indexes`
- `all_col_comments`
- `user_col_comments`
- `all_tab_comments`
- `user_tab_comments`

Commands

```
DESCRIBE dictionary;
```

```
SELECT *
FROM dictionary
WHERE ...;
```

```
COMMENT ON TABLE name
IS 'text';
```

```
COMMENT ON COLUMN table.column
IS 'text';
```

12. DCM

Privileges

- every user has username and password
- admin grants privileges to users
- DB safety
    - system safety
    - data safety
- types

    - system privileges = particular operations in DB (not assotiated with particular objects) e.g. DB access
    - object privileges = DML on particular objectname
- scheme = tables + views + sequences
- role is union of privileges

System privileges

- $>$ 100
- admin has highest-level system privs (create users, delete users, delete tables, backup tables...)
- WITH ADMIN OPTION allows user to grant his obtained system privilegy to another user

```
CREATE USER username
IDENTIFIED BY password;


CREATE ROLE rolename;


GRANT {priv1 | role0], ..., [privN]
TO {user1 | role1 | PUBLIC}, ..., [userN]
[WITH ADMIN OPTION];


ALTER USER username
[IDENTIFIED BY password]
[ACCOUNT {UNLOCK | LOCK}];
```

privileges:

- `create session` - minimal priv that allows connect to DB
- `create table`
- `create sequence`
- `create view`
- `create procedure`

*scott (one of oracle dev) has password "tiger" (his cat name)

Object privileges

- creator has all privileges on his object

- creator can grant privileges on his object
- PUBLIC = all users of system
- WITH GRANT OPTION allows user to grant his obtained object privilegy to another users

```
GRANT privilegy [(columns)]  --columns for "update"
ON [owner.]objectname
TO {users|roles|PUBLIC}
[WITH GRANT OPTION];
```

objects and their privileges:

- table
  - alter
  - delete
  - index
  - insert
  - references
  - select
  - update
- view
  - delete
  - insert
  - select
  - update
- sequence
  - alter
  - select
- procedure
  - execute

Revoke

- revokes all privs that was obtained WITH GRANT OPTION (cascade delete) but not revokes WITH ADMIN OPTION
- CASCADE CONSTRAINTS is actual with [references|ALL] = constraints will be deleted too

```
REVOKE {privs|ALL}
ON objectname
```

```
FROM {users|roles|PUBLIC}
[CASCADE CONSTRAINTS]
```

Dictionary views

- role_sys_privs - sys privs
- role_tab_privs - obj privs
- user_sys_privs
- user_role_privs - roles that users have
- user_tab_privs_made - that user grants
- user_tab_privs_recd - that user has
- user_col_privs_made
- user_col_privs_recd

13. OBJECTS 2

Alter table (columns)

- add column
- alter column datatype
- add default to column (only future inserts)
- delete column
- rename column

```
ALTER TABLE name
ADD (columnname datatype [DEFAULT expression], ...);
```

```
ALTER TABLE name
MODIFY (columnname datatype [DEFAULT expression], ...);
```

- CASCADE CONSTRAINTS delete all constrainst that refer to column (including multi-column)

```
ALTER TABLE name
DROP {COLUMN columnname|(columns)}
[CASCADE CONSTRAINTS];
```

- columns can be marked as UNUSED and deleted in future
- user_unused_col_tabs - all unused columns

```
ALTER TABLE name
SET UNUSED {COLUMN columnname|(columns)};
```

```
ALTER TABLE name
DROP UNUSED COLUMNS;
```

Alter table (constraints)

- add/drop constraint
- not modify constraint
- enable/disable constraint (???)
- validate/novalidate constraint (???)
- column/table level constraint

```
ALTER TABLE name
ADD ([CONSTRAINT name] CONSTR_TYPE (column), ...);
```

- MODIFY can be used with PK, NOT NULL:

```
ALTER TABLE name
MODIFY (column CONSTR_TYPE);
```

Deferrable constraints

- NOT DEFERRABLE - will be checked line-by-line
- DEFERRABLE - constrains will be checked after commit
    - INITIALLY DEFERRED - on when created
    - INITIALLY IMMEDIATE - off when created
- e.g. recursive FK

```
ALTER TABLE name
ADD ... CONSTR_TYPE (column)
[{
  NOT DEFERRABLE | DEFERRABLE
  [INITIALLY {DEFERRED | IMMEDIATE}]
}];
```

```
SET CONSTRAINTS name {DEFERRED | IMMEDIATE};


ALTER SESSION
SET CONSTRAINTS = {DEFERRED | IMMEDIATE};
```

- CASCADE: if PK then FK will be also deleted

```
ALTER TABLE name
DROP {CONSTRAINT constr_name | PRIMARY KEY} [CASCADE];
```

- temporary freezing of constraints
- CASCADE - disable constraints that reference to column
- when enable PK or UNIQUE then indexes are created

```
ALTER TABLE name
{DISABLE | ENABLE} CONSTRAINT constr_name
  [CASCADE];
```

Indexes

- autimatic creation when PK/UNIQUE
- handle creation when not only CREATE INDEX but also CREATE TABLE (to set a name of index manually)

```
CREATE TABLE table
(column DATATYPE
  PRIMARY KEY USING INDEX
  (CREATE INDEX name ON table(column)), ...);
```

- index on function

```
CREATE INDEX name
ON table(expression);
```

Flashback table

- recover table to early timestamp

- if PURGE then doesn't work
- SCN = system change number (integer number that corresponds to a timestamp)
- new_name if there is new table with that name already

```
FLASHBACK TABLE [scheme.]name1 [as new_name], ...
TO {timestamp | SCN | BEFORE DROP} expression
[ {ENABLE | DISABLE} TRIGGERS ];
```

- view recyclebin
- table can has new name in recyclebin

```
SELECT original_name, operation, droptime
FROM recyclebin;
```

External tables

- alias of full path to the folder
- `all_directories`

```
CREATE OR REPLACE DIRECTORY name
AS '/.../name';
```

```
GRANT {read|write} ON DIRECTORY name TO user;
```

```
GRANT create directory;
```

- driver_type - driver type of data access
  - `oracle_loader`
  - `oracle_datapump`
- dir - oracle directory object (alias)
- ACCESS PARAMETERS
- REJECT LIMIT - number of errors
- PARALLEL - number of parallel processes that oracle will launch
- RECORD DELIMITED - how strings will be delimited in file
- FIELDS TERMINATED - how columns will be delimited in file
- BADFILE - file with errors
- (column POSITION(n:m) ...) - fixed-length format
- lec 8 ~1:00:00 (empty rows must be deleted)

```
CREATE TABLE name
  (columns) --not constraints!
ORGANIZATION EXTERNAL
( TYPE driver_type
  DEFAUL DIRECTORY dir
  ACCESS PARAMETERS
  (
    RECORD DELIMITED BY {NEWLINE|'symbol'}
    NOBADFILE | BADFILE [dir:'filename']
    NOLOGFILE | LOGFILE [dir:'filename']
    FIELDS TERMINATED BY 'symbol' --'|'
    [(column POSITION(n:m) DATATYPE, ...)]
  )
LOCATION ('file_name') )
PARALLEL n=1
REJECT LIMIT {n=0 | UNLIMITED};
```

14. MANIPULATING BIG DATA

Subqueries

- copy from another table:

```
INSERT INTO table(columns)
SELECT...
```

- target subselect:
- can use CHECK OPTION

```
INSERT INTO ... SELECT...
VALUES
```

- subselect in FROM

```
SELECT ...
FROM (SELECT ...) ...;
```

- subselect in UPDATE and DELETE

```
UPDATE name
SET column = (SELECT... WHERE ...) ...
WHERE ...;


DELETE name
WHERE column = (SELECT... WHERE ...);
```

Default

```
INSERT INTO name
 (columns)
VALUES (values|DEFAULT);


UPDATE name
SET columns = DEFAULT
[WHERE ...];
```

Multi-table insertion

- data storage supporting systems
- SELECT is source always
- values may be different in tables

unconditional insert:

```
INSERT ALL
  INTO table1 [(columns)] VALUES(src_values)
  [INTO table1 ... VALUES(another_src_values)]
  ...
  [INTO table2 ...]
  ...
(SELECT src_values
FROM sourcetab
WHERE ...);


INSERT ALL
  INTO table1 [(columns)] VALUES(values)
  [INTO table1 ... VALUES(another_values)]
```

```
  ...
(SELECT 1 --any n, doesn't matter
FROM dual);
```

conditional insert:

- ALL - all rows will be checked for all conditions
- FIRST - if row satisfies the condition then next conditions will be missed

```
INSERT {ALL | FIRST}
WHEN cond1 THEN
  INTO table1 [(columns)] VALUES(values)
WHEN cond2 THEN
  INTO table2 ...
  ...
(SELECT values
FROM sourcetab
WHERE ...);
```

Merge

- insert + update + delete = merge
- big efficiency and easy of use

```
MERGE INTO table [tab_al]
USING {tab|view|(sub)} [al] --if sub alias necessary
ON (cond)
WHEN MATCHED THEN
  UPDATE SET
  [tab_al.]col1 = [al.]src_val1
  [tab_al.]col2 = [al.]src_val2
  ...
WHEN NOT MATCHED THEN
  INSERT [([tab_al.]cols)]
  VALUES (al.src_vals)


MERGE INTO ...
WHEN MATCHED THEN
  UPDATE SET ... [WHERE cond1] --ref on src and rcv
DELETE WHERE cond2 --ref ob src
WHEN NOT MATCHED THEN
  INSERT ... [WHERE cond3]
  [LOG ERRORS
```

```
    [INTO [scheme.]table_log]
    [(simple_expr)]
    [REJECT LIMIT {n=0|UNLIMITED}]]
```

- table for errors

```
EXEC dbms_errlog.create_error_log (table, table_log);
```

```
SELECT *
FROM table_log;
```

Tracking changes in data

- Flashback Version Query

```
SELECT salary FROM eployees3
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 107;
```

*????

```
SELECT salary FROM eployees3
VERSIONS STARTTIME and VERSION ENDTIME
WHERE employee_id = 107;
```

15. GROUPING WITH RELATED DATA

ROLLUP and CUBE

```
SELECT...
FROM ... [WHERE ...]
[GROUP BY ...,
  [ROLLUP(col1, ..., colN)],
  [CUBE(col1, ..., colN)]]
[ORDER BY ...]
```

examples:

- `ROLLUP(a, b, c)` = result for a+b+c, a+b, a, general
- `CUBE(a, b, c)` = result for a+b+c, a+b, a+c, b+c, a, b, c, general (2^n results, n - number of columns)

GROUPING

```
SELECT ..., GROUPING(col)
...
```

- 1 if empty value from ROLLUP or CUBE, 0 if another
- only one column in argument

GROUPING SETS

- several groupings in one select instead of several selects and union all
- effective if:
  - efficiency increase with number of elements in grouping sets
  - short select (without unions)
  - one pass of base table
- composite column `(a, b)` is processed as single unit

```
SELECT...
FROM ... [WHERE ...]
[GROUP BY ...,
  [GROUPING SETS((col, ..., (cols))] ]
[ORDER BY ...]
```

exmples:

- `GROPING SETS ((a, b), (b, c))` - result for a+b, b+c
- `GROPING SETS ((a, b, c), ())` - result for a+b+c, general

Linked Groupings

- `GROUP BY`

16. TIME ZONES

session parameters

absolute offset:

```
ALTER SESSION
SET TIME_ZONE = '-05:00';
```

time zone of DB:

```
ALTER SESSION
SET TIME_ZONE = DBTIMEZONE;
```

time zone of OS:

```
ALTER SESSION
SET TIME_ZONE = LOCAL;
```

setting the time zone by name:

```
ALTER SESSION
SET TIME_ZONE = 'America/New_York';
```

alter date format in session:

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
```

time zone functions

`SESSIONTIMEZONE` - current time zone
`CURRENT_DATE` - current date
`CURRENT_TIMESTAMP` - current timestamp with time zone
`LOCALTIMESTAMP` - ... without time zone
`DBTIMEZONE` - time zone of DB
`SESSIONTIMEZONE` - time zone of OS

```
SELECT SESSIONTIMEZONE, CURRENT_DATE,
  CURRENT_TIMESTAMP, LOCALTIMESTAMP,
  DBTIMEZONE, SESSIONTIMEZONE
FROM dual;
```

timestamp datatype

- = `DATE` but with fractions of second
- types:
  - `TIMESTAMP[(precision_in_fracs_of_sec)]`
  - `TIMESTAMP[(...)] WITH TIME ZONE` - with time zone offset in hours and minutes
  - `TIMESTAMP[(...)] WITH LOCAL TIME ZONE` - time value of time consider time zone
- values:
  - `YEAR` : -4712 to 9999 excluding 0
  - `MONTH` : 01 to 12
  - `DAY` : 01 to 31
  - `HOUR` : 00 to 23
  - `MINUTE` : 00 to 59
  - `SECOND` : 00 to 59.9(N) where 9(N) is precision, N=6 by default
  - `TIMEZONE_HOUR` : -12 to 14
  - `TIMEZONE_MINUTE` : 00 to 59
  -

interval datatype

- types:
  - `INTERVAL YEAR[(year_prec)] TO MONTH`
  - `INTERVAL DAY[(day_prec)] TO SECOND`
- precision depends on fact of fields in interval and on the qualifier that user sets
- fields:
  - `YEAR` : any Z
  - `MONTH` : 00 to 11
  - `DAY` : any Z
  - `HOUR` : 00 to 23
  - `MINUTE` : 00 to 59
  - `SECOND` : 00 to 59.9(N) where 9(N) is precision

extract and convert functions

- `EXTRACT(field FROM date)` - some field from date datatype
- `TZ_OFFSET('timezone_name')` - offset for timezone
- convert timestamp to timestamp with timezone
  - `FROM_TZ(TIMESTAMP 'value_str', 'timezone_offset')`
  - `FROM_TZ(TIMESTAMP 'value_str', 'timezone_name')`
- convert string to timestamp or timestamp with timezone:
  - `TO_TIMESTAMP(str)`
  - `TO_TIMESTAMP_TZ(str)`
- convert string to interval:
  - `TO_YMINTERVAL(str)`
  - `TO_DSINTERVAL(str)`

17. SUBQUERIES 2

multi-column subqueries

every column will be compared with the corresponding column in subquery (paired comparison):

```
SELECT ...
FROM ...
WHERE (col_11, col_12) IN (
  SELECT col_21, col_22
  FROM ...
  WHERE ...
);
```

every column will be compared with each column (unpaired comparison):

```
SELECT ...
FROM ...
WHERE col_11 IN (
  SELECT col_21
  FROM ...
  WHERE ...
)
```

```
AND col_12 IN (
  SELECT col_22
  FROM ...
  WHERE ...
);
```

scalar subquery

- returns one value from one column and string
- can be used in DECODE, CASE and every SELECT sentence except for GROUP BY

relation subquery

- line-by-line processing
- one subquery for each row of query (low efficiency)
- get->execute->use

```
SELECT col_1, ...
FROM outer
WHERE col_1 > (
  SELECT col_2
  FROM inner
  WHERE expr_1 = outer.expr_2
);
```

- `EXISTS` - if subquery has 1+ rows then TRUE else FALSE
- `NOT EXISTS`

```
SELECT ...
FROM ...
WHERE EXISTS (relation_subquery);
```

- update with relation subquery

```
UPDATE tab_1
SET col = (
  SELECT expr
  FROM tab_2
  WHERE tab_1.col_1 = tab_2.col_2
);
```

- delete with relation subquery

```
DELETE FROM tab_1
WHERE col > (
  SELECT expr
  FROM tab_2
  WHERE tab_1.col_1 = tab_2.col_2
)
```

with

```
WITH
alias_1 AS (
  SELECT ...
),
alias_2 AS (
  SELECT ...
)
SELECT ... <using_alias_1_and_2_several_times>;
```

18. HIERARCHICAL DATA

```
SELECT [LEVEL], ...
FROM ...
[WHERE ...]
[START WITH condition]
[CONNECT BY [NOCYCLE] {PRIOR|LEVEL} condition]
[ORDER [SIBLINGS] BY ...] ...;
```

- sections `START WITH` and `CONNECT BY` can follow in any order
- if `WHERE` has join operation then join is performed first and `CONNECT BY` - second, otherwise `CONNECT BY` is first
- `START WITH` define the starting point; solutions will be builded from each possible points if it is not specified
- `CONNECT BY` define relation between parent and child rows in hierarchy:
  - `CONNECT BY PRIOR parent = child` - *top-down* crawl
  - `CONNECT BY PRIOR child = parent` - *bottom-up* crawl

    - can contain another conditions-filters
    - can **NOT** cotain *subquery*
    - return error if cause *loop*
    - `PRIOR` returns previous link (`NULL` if there is no previous); it can be used in `SELECT`, `WHERE`, `CONNECT BY` and `ORDER BY`
- `LEVEL` returns nuber of level in hierarchy from the starting of crawl
- `NOCYCLE` fix cycle error

skip a node in tree (his parent will be connected with his child):

```
SELECT ...
FROM ...
WHERE col != node_val
```

skip a branch in tree (all branch that begin from this node will be deleted):

```
SELECT ...
FROM ...
CONNECT BY PRIOR col1=col2
  AND col != node_val
```

- `SYS_CONNECT_BY_PATH(col, char)` returns path from the root to the node with separator (char) between links
- `CONNECT_BY_ISLEAF` returns 1 if current node doesn't have children, else 0
- `CONNECT_BY_ROOT` returns root
- `CONNECT_BY_ISCYCLE` returns 1 if cause cycle, else 0 (works only with `NOCYCLE`)
- `SIBLINGS` sorts first by level and second by ASC

organizing 1-10 cycle using `CONNECT BY LEVEL`:

```
SELECT LEVEL
FROM DUAL
CONNECT BY LEVEL <= 10;
```

organizing 1-10 cycle using `WITH`:

```
WITH
numbers(n) AS (
    SELECT 1 n
    FROM dual
    UNION ALL
    SELECT n+1 n
    FROM numbers
```

```
    WHERE n < 10
)
SELECT n
FROM numbers;
```

19. REGULAR EXPRESSIONS

metacharacters

^ - match with the beginning of str or logically NOT if on first position in [^...]
$ - with the ending of str
. - with any symbol except of **newline** symbol
| - logical OR
[ ] - list of expressions and match with anyone of them
{m} - match exactly m times
{m,n} - match m-n times
{m,} - match m+ times
* - 0+ repetitions
+ - 1+ repetition
? - 0-1 repetition
( ) - expression will be processed as solid subexpression
[: :] - character class and match with any of them
\ - just symbol or servis character (starting of another symbol or operator)
[= =] - equivalence of classes

- for [=e=]:
- NLS_SORT russian: e, E
- NLS_SORT binary: e, E, ë, Ë
- NLS_SORT french: e

\n - backlink to expression in () where n is 1,2,...
[. .] - for example multi-character element for compare

character classes

[:lower:] - lowercase alphabetic
[:upper:] - uppercase alphabetic
[:alpha:] - [:lower:] U [:upper:]

[:digit:] - digits
[:alnum:] - [:alpha:] U [:digit:]
[:punct:] - punctuaction
[:graph:] - [:alnum:] U [:punct:]
[:blank:] - space U tab
[:space:] - [:blank:] U carriage return U page break
[:print:] - printable
[:cntrl:] - unprintable U control
[:xdigit:] - hexademical

perl

\d - [[:digit:]]
\D - [^[:digit:]]
\w - [[:alnum:]_]
\W - [^[:alnum:]_]
\s - [[:space:]]
\S - [^[:space:]]
\A - match only in the beginning of the str
\z - match only in the ending of the str
\Z - \z or before new line character in multiline text

functions

- `REGEXP_LIKE(str, pattern, [match_op='c'])` - searches (like `LIKE` but with regular expressions)
- `REGEXP_REPLACE(str, pattern, [replace_str, [start_pos=1, [match_num=1, [match_op='c']]]])` searches and replaces
- `REGEXP_INSTR(str, pattern, [start_pos=1, [match_num=1, [return_op=0, [match_op='c']]]])` searches and returns position
- `REGEXP_SUBSTR(str, pattern, [start_pos=1, [match_num=1, [match_op='c']]])` searches and returns substring
- `REGEXP_COUNT(str, pattern, [start_pos=1, [match_op='c']])` searches and returns number of matches

return_op: if 0 then pos of first char of pattern, if 1 then pos of first char after pattern

match_options:

- `'c'` - case sensitive

- `'i'` - case insensitive
- `'n'` - then '.' include new line character
- `'m'` - input is multi-line string
- `'x'` - ignore spaces
- it's possible to use several match options ('imn')
- options are only in lowercase

20. ANALYTIC FUNCTIONS

common

types:

- window (sectional)
- ranking
- summary
- statistical

notes:

- syntax constructions should follow in definite order
- can be used only in `SELECT` and `ORDER BY` (not in `WHERE`)

syntax

```
FUNC_NAME( [ <args> ] ) OVER (
  [ <query_partition_clause> ]
  [ <order_by_clause>
  [ <windowing_clause> ]]
  )
```

- `OVER` - keyword for analitical functions
- `<query_partition_clause>` - `GROUP BY` analog (fragmentation)
  - one big collection by default
- `<order_by_clause>` - `ORDER BY` analog
  - not for result but for calculations
  - necessary if <windowing_clause>
- `<windowing_clause>` - moving or rigid window (collection/interval) of data for work

- cannot be defined if no <order_by_clause>

<query_partition_clause>:

```
PARTITION BY col
```

<order_by_clause>:

```
ORDER BY col
```

<windowing_clause>:

```
{ROWS|RANGE}
BETWEEN
  {UNBOUNDED PRECEIDING | CURRENT ROW | expr PRECEIDING} --lower (previous) bound
  AND
  {UNBOUNDED FOLLOWING | CURRENT ROW | expr FOLLOWING} --upper (next) bound
```

- `RANGE` - define window by values range
- `ROWS` - define window by offset relative to the current line
- `UNBOUNDED PRECEIDING` - from 1st row
- `CURRENT ROW` - from current row
- `BETWEEN` and upper bound aren't necessary
- by default:

```
RANGE
BETWEEN
UNBOUNDED PRECEIDING
AND
CURRENT ROW
```

window (sectional)

*with intermediate values as opposed to simple aggregate functions

- `SUM`
- `MAX`
- `MIN`
- `AVG`
- `COUNT`
  - without NULLs
  - COUNT(*) - all values

ranking

- `ROW_NUMBER` - calculate row number
- `RANK`
  - if last value is equal current value then same number
  - else number is equal number of row
- `DENSE_RANK`
  - if last value is equal current value then same number
  - else number is last number + 1
- `NTILE(n)` - divide by n fragments
  - if (count(str)%n=0) then will be n groups with (count(str)/n) strings (numbers 1-n will be assigned to this groups)
  - if (count(str)%n!=0) then remainder of division will be evenly distribute into groups (distribution starts with group with smaller number)

summary

*like window but without <order_by_clause> and consequently show common results for each row of group

- `SUM`
- `MAX`
- `MIN`
- `AVG`
- `COUNT`
- `LAG(col, [n=1], [default=NULL])` - refer to previous (upper) row with offset n
  - `default` is value that will be used if row doesn't exist. NULL by default
- `LEAD(col, [n=1], [default=NULL])` - refer to next (lower) row with offset n

- ```
  LISTAGG(expr, [divider=NULL])
  WITHIN GROUP (ORDER BY col)
  ```

  - concatenate expressions with divider as string
  - NULL values will be ignored
  - `WITHIN GROUP` sorts values of expression
  - if it's analytic function then `GROUP BY` in `SELECT` is prohibited
  ```

statistical

- `CORR(expr1, expr2)` - correlation coefficient
- `COVAR_POP(expr1, expr2)` - covariance of the aggregate of a pair of expressions
- `COVAR_SAMP(expr1, expr2)` - selective covariance of a pair of expressions
- `VAR_POP(exprs)` - variance of the aggregate of a pair of expressions
- `VAR_SAMP(exprs)` - selective variance of a pair of expressions
- `VARIANCE([DISTINCT|ALL] expr)` - variance of expression

21. MODEL

common

- only in oracle
- since 10g version
- can't change tables
- before `ORDER BY`

```
SELECT ...
FROM ...
MODEL
DIMENSION BY (<columns>|<expressions>|<const>)
MEASURES (<columns>)
RULES ([<rules>])
```

- `DIMENSION BY` - columns that define cells of "array" uniquely
- `MEASURES` - **ALL** another columns from `SELECT` or even others
- `RULES` - rules of calculations
  - references:
    - single-cell positional: `column[val]`
    - single-cell named: `column[col = val]`
    - multi-cell positonal: `column[ANY]`
    - multi-cell named: `column[col BETWEEN a AND b]`
  - first level nested references are allowed: `column[col[val] = value]`
  - references as rvalue when it's a res of group function above range (scalar value)
  - 0 rules are allowed
  - loops and cases

- `CV([])` - current value of column from `DIMENSION BY`

other

```
SELECT ...
FROM ...
MODEL

[RETURN {ALL|UPDATED} ROWS]
[MAIN model_name]
[PARTITION BY (<columns>)]

DIMENSION BY (<columns>|<expressions>|<const>)
MEASURES (<columns>)

[IGNORE NAV | KEEP NAV]
[UNIQUE DIMENSION | UNIQUE SINGLE REFERENCE]

RULES

  [UPDATE | UPSERT | UPSERT ALL]
  [AUTOMATIC ORDER | SEQUENTIAL ORDER]
  [ITERATE(number) [UNTIL cond]]]

  ([<rules>])
```

- `RETURN {ALL|UPDATED} ROWS` - ALL by default, return all or updated rows
- `MAIN model_name` - name of model
- `PARTITION BY (<columns>)` - if not specified that 1 big group
- `IGNORE NAV | KEEP NAV` - nav is null
  - keep nav by default
  - ignore nav replace null with default values (0, 01.01.2001, '')
- `UNIQUE DIMENSION | UNIQUE SINGLE REFERENCE` - key in model
  - unique dimension by default
  - unique single reference allow to have duplicates of key values
    - you can set rules for all rows: `col[ANY]=val`
    - or limit left part of rule: `col[non_unique_val]=val`
    - but you can't use it as rvalue
- `AUTOMATIC ORDER | SEQUENTIAL ORDER` - order of rules executing
  - automatic - with accounting of logical dependencies between rules

    - sequential - (by default) rules will be executed in order of recording
- `ITERATE(number) [UNTIL cond]` - used with loops, only when `SEQUENTIAL ORDER`
    - number - count of repetition (from 0 to number)
    - cond - for exit from loop
- `UPDATE | UPSERT | UPSERT ALL`
    - update - only update elements
    - upsert - (by default) update & insert elements with positional references
    - upsert all - work like upsert but when there is mix of positional and symbolic references then it:
        - find all rows that match with symbolic reference
        - make cross join with positional references
        - insert all non-existed rows from there
    - this options can be specified before one of rules in brackets or before brackets (for all rules simultaneously)

```
PRESENTV(cell, expr1, expr2)
```

- use only in `MODEL` and only as rvalue in `RULES`
- returns expr1 when cell exists in input data, else returns expr2

EXTRA

```
CAST({col|const|fun} AS DATATYPE)
```