MTN.BI.02

ORACLE SQL FOUNDATION

Oracle Data types, Functions and Sub-queries

Author: **Aliaksandr Chaika**
Senior Software Engineer
Certified Oracle Database SQL Expert
Aliaksandr_Chaika@epam.com

Objectives

1. Oracle Data Types
2. Functions
3. Subqueries

DATA TYPES

ANSI Data Types

SQL supports three sorts of data types:

1. **predefined data types**
2. **constructed types**
3. **user-defined types**

Oracle supports constructed (reference, rowtype, collection) and user-defined types. These constructions mostly used for PL/SQL programming.

There is no TIME equivalents in Oracle.
BOOLEAN is allowed in PL/SQL only.

ANSI Predefined data types:

1. CHARACTER
2. CHARACTER VARYING
3. CHARACTER LARGE OBJECT
4. BINARY
5. BINARY VARYING
6. BINARY LARGE OBJECT
7. NUMERIC
8. DECIMAL
9. SMALLINT
10. INTEGER
11. BIGINT
12. FLOAT
13. REAL
14. DOUBLE PRECISION
15. BOOLEAN
16. DATE
17. TIME
18. TIMESTAMP
19. INTERVAL

Oracle Data Types

Oracle data types:

1. **Oracle Built-in Types**
2. **ANSI, DB2, and SQL/DS Data Types**
3. **User-Defined Types**
4. **Oracle-Supplied Types**

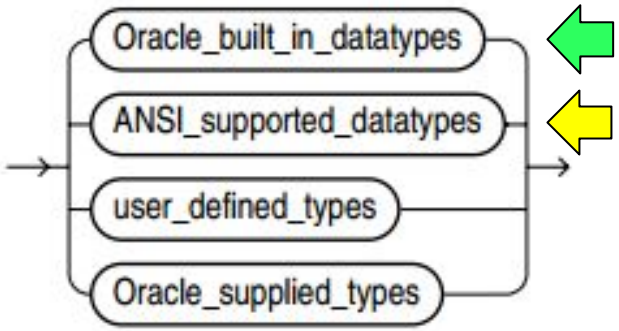
Oracle built-in data types:

1. CHAR[(size[BYTE| CHAR])] — Character
2. NCHAR[(size[BYTE| CHAR])] — Character
3. VARCHAR2(size[BYTE| CHAR]) — Character
4. NVARCHAR2(size) — Character
5. NUMBER [(precision[, scale])] — Numeric
6. FLOAT[(precision)] — Numeric
7. BINARY_FLOAT — Numeric
8. BINARY_DOUBLE — Numeric
9. DATE — Date / Timestamp / Interval
10. TIMESTAMP[(fractional_seconds_precision)] — Date / Timestamp / Interval
11. TIMESTAMP[(fractional_seconds)] WITH TIME ZONE — Date / Timestamp / Interval
12. TIMESTAMP[(fractional_seconds)] WITH LOCAL TIME ZONE — Date / Timestamp / Interval
13. INTERVAL YEAR[(year_precision)] TO MONTH — Date / Timestamp / Interval
14. INTERVAL DAY[(day_precision)] TO SECOND[(fract_sec)] — Date / Timestamp / Interval
15. CLOB — LOB
16. NCLOB — LOB
17. BLOB — LOB
18. BFILE — LOB
19. LONG — LOB
20. RAW(size) — Raw data
21. LONG RAW — Raw data
22. ROWID — Raw data
23. UROWID[(size)] — Rowid

The LONG is legacy data type and provided for backward compatibility only. Only one LONG field can be in a table.

Oracle Data Types Definitions

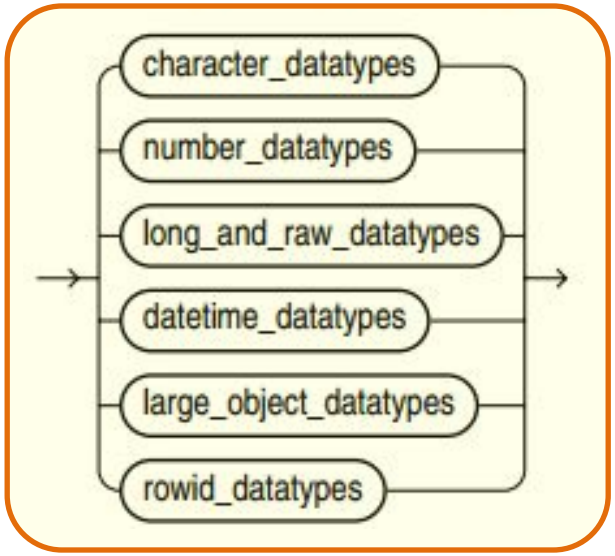
datatypes::=



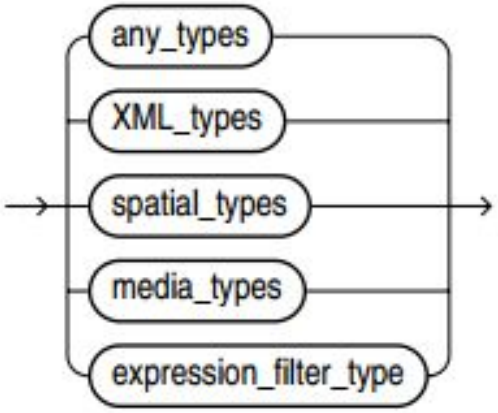
Oracle's native data types

ANSI data types which not contradictory to native (except TIME and BOOLEAN)

Oracle_built_in_datatypes::=

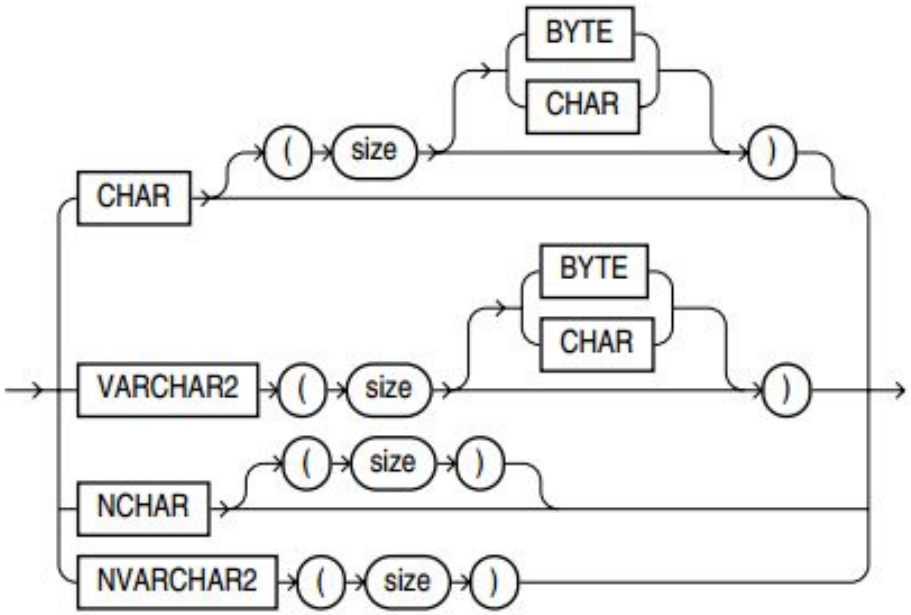


Oracle_supplied_types::=



Character Data Types

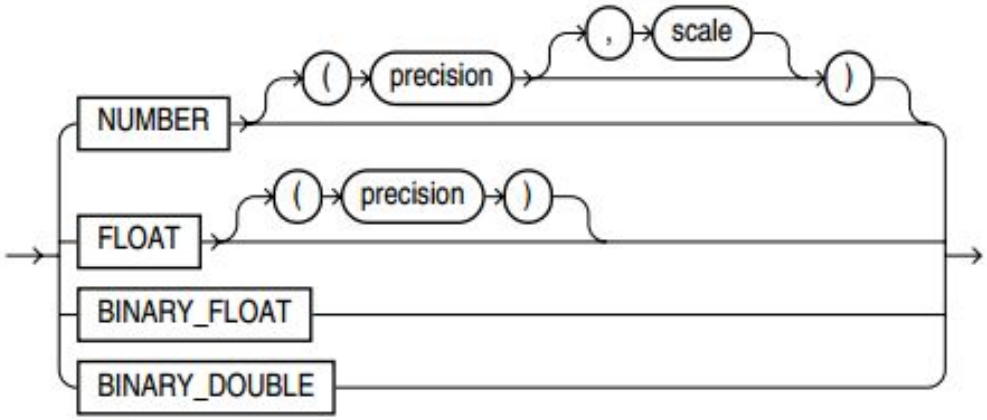
character_datatypes::=



CHAR [(size [BYTE|CHAR])]	Fixed-length character data of length size bytes or characters. Maximum size is 2000 bytes or characters. Default and minimum size is 1 byte. BYTE indicates that the column will have byte length semantics. CHAR indicates that the column will have character semantics.
NCHAR [(size [BYTE|CHAR])]	Fixed-length character data of length size characters. The number of bytes can be up to two times size for AL16UTF16 encoding and three times size for UTF8encoding. Maximum size is determined by the national character set definition, with an upper limit of 2000 bytes. Default and minimum size is 1 character.
VARCHAR2 (size [BYTE|CHAR])	Variable-length character string having maximum length size bytes or characters. Maximum size is 4000 bytes or characters, and minimum is 1 byte or 1 character.
NVARCHAR2 (size)	Variable-length Unicode character string having maximum length size characters. The number of bytes can be up to two times size for AL16UTF16 encoding and three times size for UTF8 encoding. Maximum size is determined by the national character set definition, with an

Numeric Data Types

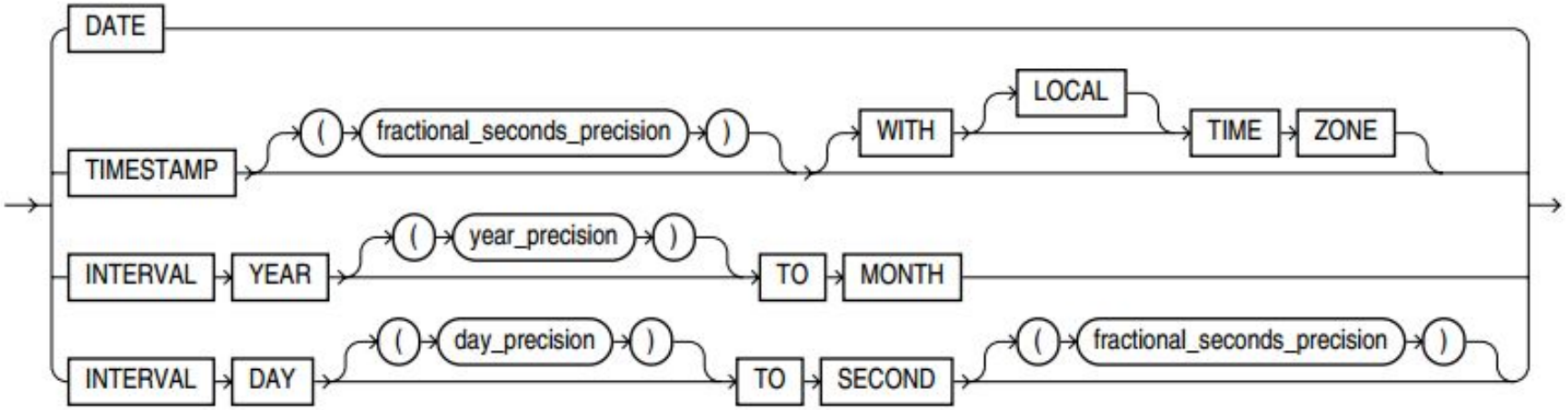
number_datatypes::=



NUMBER [(p[, s])]	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127. Both precision and scale are in decimal digits. A NUMBER value requires from 1 to 22 bytes.
FLOAT[(precision)]	A subtype of the NUMBER data type having precision p. A FLOAT value is represented internally as NUMBER. The precision p can range from 1 to 126 binary digits. A FLOAT value requires from 1 to 22 bytes.
BINARY_FLOAT	32-bit floating point number. This data type requires 4 bytes.
BINARY_DOUBLE	64-bit floating point number. This data type

Datetime Data Types

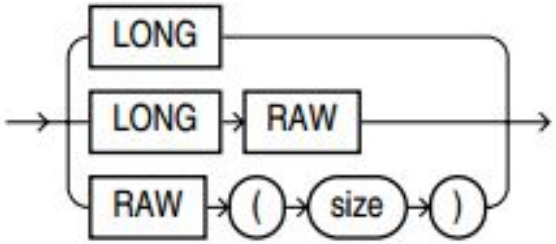
datetime_datatypes::=



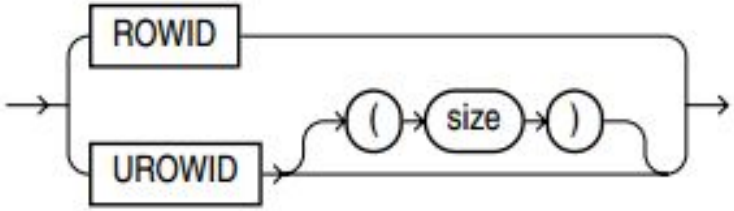
DATE	The size is fixed at 7 bytes. This data type contains the date and time fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.
TIMESTAMP [(fractional_seconds_precision)]	DATE + fractional seconds fractional_seconds_precision is from 0 to 9. The default is 6.
TIMESTAMP[(fractional_seconds)] WITH TIME ZONE	All values of TIMESTAMP as well as time zone displacement value.
TIMESTAMP[(fractional_seconds)] WITH LOCAL TIME ZONE	All values of TIMESTAMP WITH TIME ZONE, with the following exceptions: • Data is normalized to the database time zone when it is stored in DB. • When the data is retrieved, users see the data in the session time zone.
INTERVAL YEAR[(year_precision)] TO MONTH	Stores a period of time in years and months, where year_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default is 2.
INTERVAL DAY[(day_precision)] TO SECOND[(fract_sec)]	Stores a period of time in days, hours, minutes, and seconds, where: • day_precisionis the maximum number DAY, from 0 to 9. The default is 2. • fractional_seconds_precision is SECODE fraction 0 to 9. The default is 6.

Large Objects (LOB), ROWID and Raw Data Types

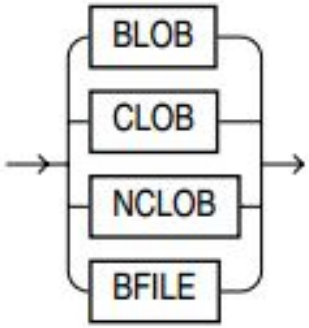
long_and_raw_datatypes::=



rowid_datatypes::=

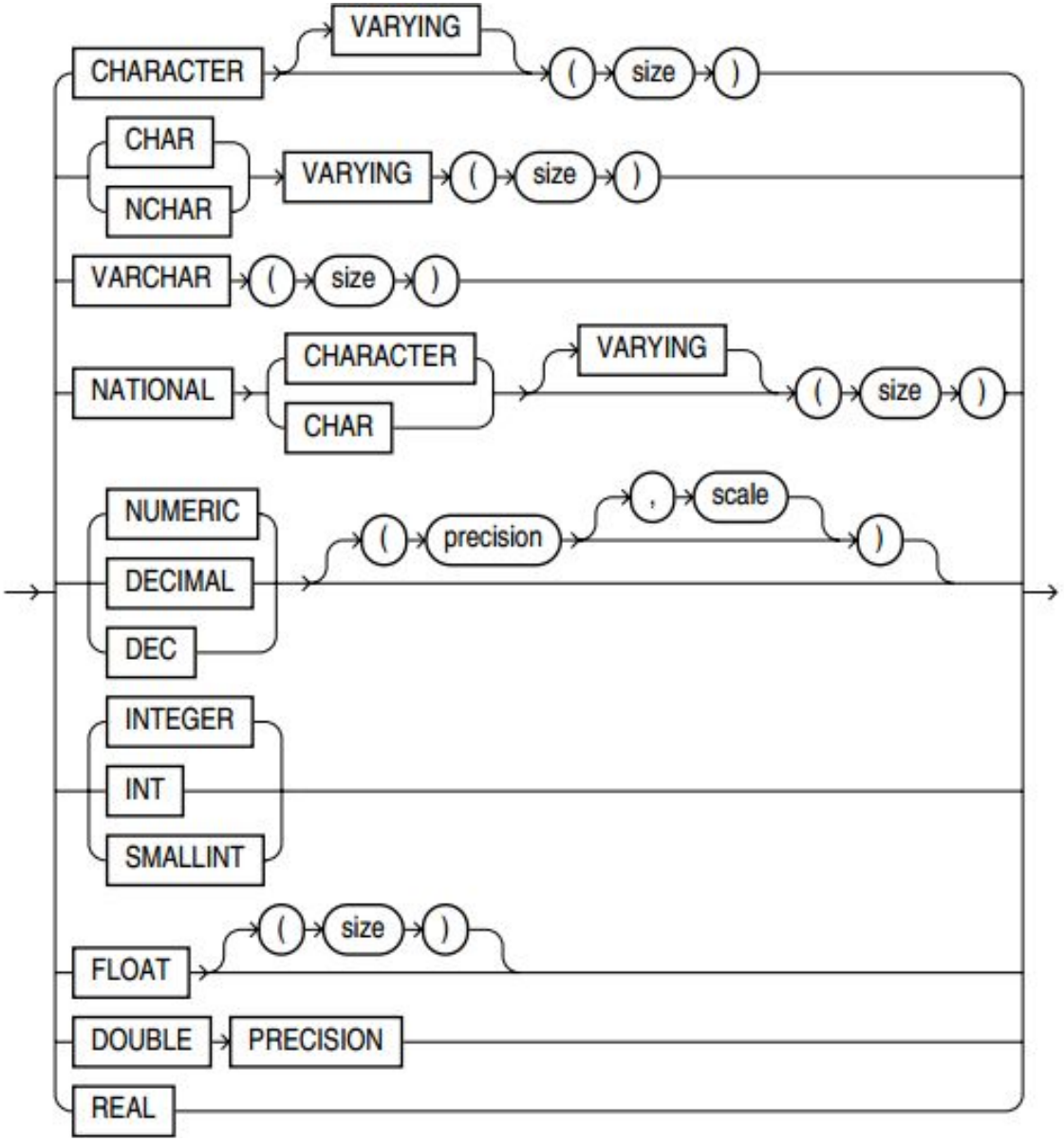


large_object_datatypes::=



CLOB	A character large object containing single-byte or multibyte characters. Both fixed-width and variable-width character sets are supported, both using the database character set. Maximum size is (4 gigabytes – 1 byte) * (database block size).
NCLOB	A character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 gigabytes – 1 byte) * (database block size). Stores national character set data.
BLOB	A binary large object. Maximum size is (4 gigabytes – 1 byte) * (database block size).
BFILE	Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes.
LONG	Character data of variable length up to 2 gigabytes. Provided for backward compatibility.

Oracle ANSI Supported Data Types



ANSI Predefined data types:

1. CHARACTER
2. CHARACTER VARYING
3. CHARACTER LARGE OBJECT
4. BINARY
5. BINARY VARYING
6. BINARY LARGE OBJECT
7. NUMERIC
8. DECIMAL
9. SMALLINT
10. INTEGER
11. BIGINT
12. FLOAT
13. REAL
14. DOUBLE PRECISION
15. BOOLEAN
16. DATE
17. TIME
18. TIMESTAMP
19. INTERVAL

ANSI Data Types in Oracle

ANSI SQL Data Type	Oracle Data Type
CHARACTER(n) CHAR(n)	CHAR(n)
CHARACTER VARYING(n) CHAR VARYING(n)	VARCHAR2(n)
NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	NCHAR(n)
NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	NVARCHAR2(n)
NUMERIC[(p,s)] DECIMAL[(p,s)]	NUMBER(p, s)
INTEGER INT SMALLINT	NUMBER(38)
FLOAT DOUBLE PRECISION REAL	FLOAT(126) FLOAT(126) FLOAT(63)

Internal Representation of ANSI Data Types in Oracle

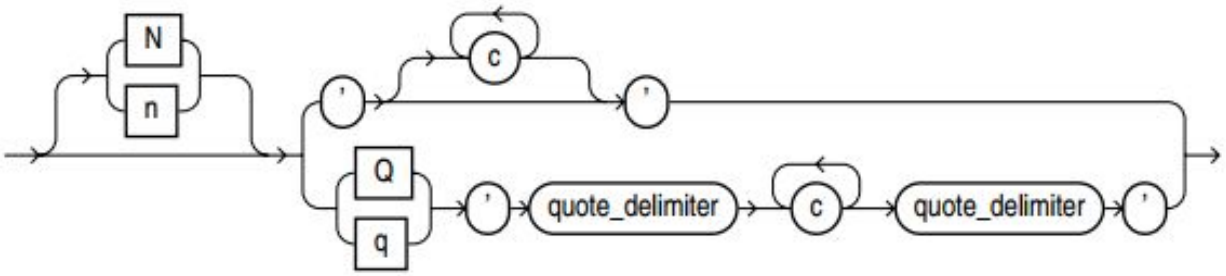
```
CREATE TABLE test_ansi_data_types  (
  CHARACTER1 CHARACTER(10),
  CHAR1 CHAR(10),
  CHARACTER_VARYING
    CHARACTER VARYING(10),
  CHAR_VARYING CHAR VARYING(10),
  NATIONAL_CHARACTER
    NATIONAL CHARACTER(10),
  NATIONAL_CHAR NATIONAL CHAR(10),
  NCHAR1 NCHAR(10),
  NATIONAL_CHARACTER_VARYING
    NATIONAL CHARACTER VARYING(10),
  NATIONAL_CHAR_VARYING
    NATIONAL CHAR VARYING(10),
  NCHAR_VARYING NCHAR VARYING(10),
  NUMERIC1 NUMERIC(5,2),
  DECIMAL1 DECIMAL(5,2),
  INTEGER1 INTEGER,
  INT1 INT,
  SMALLINT1 SMALLINT,
  FLOAT1 FLOAT,
  DOUBLE_PRECISION DOUBLE PRECISION,
  REAL1 REAL
);
DESC test_ansi_data_types
DROP TABLE test_ansi_data_types ;
```

```
table TEST_ANSI_DATA_TYPES created.
DESC test_ansi_data_types
Name                          Null Type
----------------------------- ---- -------------
CHARACTER1                         CHAR(10)
CHAR1                              CHAR(10)
CHARACTER_VARYING                  VARCHAR2(10)
CHAR_VARYING                       VARCHAR2(10)
NATIONAL_CHARACTER                 NCHAR(10)
NATIONAL_CHAR                      NCHAR(10)
NCHAR1                             NCHAR(10)
NATIONAL_CHARACTER_VARYING         NVARCHAR2(10)
NATIONAL_CHAR_VARYING              NVARCHAR2(10)
NCHAR_VARYING                      NVARCHAR2(10)
NUMERIC1                           NUMBER(5,2)
DECIMAL1                           NUMBER(5,2)
INTEGER1                           NUMBER(38)
INT1                               NUMBER(38)
SMALLINT1                          NUMBER(38)
FLOAT1                             FLOAT(126)
DOUBLE_PRECISION                   FLOAT(126)
REAL1                              FLOAT(63)

table TEST_ANSI_DATA_TYPES dropped.
```

ANSY data types have been converted to Oracle native data types

Character Data Type Literals and Comparisons



Character values are compared on the basis of two measures:

1. **Binary or linguistic sorting**
2. **Blank-padded or nonpadded comparison semantics**

```
-- TRUE (blank-padded
comparison)
SELECT CASE WHEN 'a ' = 'a  '
  THEN 'TRUE'
  ELSE 'FALSE'
END
FROM dual;

-- FALSE (empty string is NULL)
SELECT CASE WHEN '' = '  '
  THEN 'TRUE'
  ELSE 'FALSE'
END
FROM dual;

-- TRUE (blank-padded
comparison)
SELECT CASE WHEN USER = 'HR  '
  THEN 'TRUE'
  ELSE 'FALSE'
END
FROM dual;
```

```
SELECT *
FROM countries
WHERE country_id > 'UK';

SELECT *
FROM countries
WHERE country_id IN ('US', 'ZM', 'ZW');
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	US	United States of America	2
2	ZM	Zambia	4
3	ZW	Zimbabwe	4

Empty String is NULL in Oracle

What are the results of these queries?

```
SELECT CASE WHEN '' = NULL
  THEN 'TRUE'
  ELSE 'FALSE'
END
FROM dual;
```

```
SELECT CASE WHEN '' IS NULL
  THEN 'TRUE'
  ELSE 'FALSE'
END
FROM dual;
```

Which query produces this result?

```
SELECT country_id, city, state_province
  FROM locations
WHERE state_province = '';
```

```
SELECT country_id, city, state_province
FROM locations
WHERE state_province IS NULL;
```

	COUNTRY_ID	CITY	STATE_PROVINCE
1	IT	Roma	(null)
2	IT	Venice	(null)
3	JP	Hiroshima	(null)
4	CN	Beijing	(null)
5	SG	Singapore	(null)
6	UK	London	(null)

Any expression containing a null always evaluates to null.

Except concatenation!

```
SELECT 'ABC' || '' || 'DEF',
  concat('ABC', concat('', 'DEF')),
  'ABC' || NULL || 'DEF',
  concat('ABC', concat(NULL, 'DEF'))
FROM dual;
```

	'ABC'||''||'DEF'	CONCAT('ABC',CONCAT('','DEF'))	'ABC'||NULL||'DEF'	CONCAT('ABC',CONCAT(NULL,'DEF'))
1	ABCDEF	ABCDEF	ABCDEF	ABCDEF

Blank-padded and Nonpadded Comparison Semantics

```
-- Nonpadded comparison (last_name is VARCHAR2)
SELECT employee_id, first_name, last_name
FROM employees
WHERE last_name = 'King  ';
```

<empty dataset>

```
SELECT employee_id, first_name, last_name
FROM employees
WHERE
  CAST(last_name AS CHAR(20)) = 'King    ';
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	100	Steven	King
2	156	Janette	King

```
-- Blank-padded comparison (country_id is CHAR)
SELECT country_id, country_name, region_id
FROM countries
WHERE country_id = 'US  ';
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	US	United States of America	2

```
SELECT country_id, country_name, region_id
FROM countries
WHERE cast(country_id as varchar2(2)) = 'US  ';
```

<empty dataset>

```
SELECT country_id, country_name, region_id
FROM countries
WHERE country_id = cast('US  ' as varchar2(10));
```

<empty dataset>

ORACLE FUNCTIONS

Oracle Functions

Function definition

- A function is a subprogram that returns a single value (or “result”) based of its arguments values.
- A function is subroutine used to encapsulate frequently performed logic. Any code that must perform the logic incorporated in a function can call the function rather than having to repeat all of the function logic.

Function may operate on zero, one, two, or more arguments:

```
Function_name
Function_name(argument, argument, ...)
```

A function without any arguments is similar to a pseudocolumn.

However, a pseudocolumn typically returns a different value for each row in the result set, whereas a function without any arguments typically returns the same value for each row.

Functions Classification

1. **Built-in functions.** Operate as defined in the Oracle SQL Language Reference and cannot be modified.
2. **User-defined functions.** Allow you to define your own logic, implemented as block of PL/SQL code.

Oracle Functions

1. **Single-Row (Scalar) Functions**
   SQL scalar functions return a single value, based on the input value(s).
2. **Aggregate Functions**
   SQL aggregate functions return a single value, calculated from values in a column.
3. **Analytic Functions**
   Analytic functions compute an aggregate value based on a group of rows.
   They differ from aggregate functions in that they return multiple rows for each group.
4. **Object Reference Functions**
   Object reference functions manipulate REF values (references to objects).
5. **Model Functions**
   Facilitate SQL for Modeling operations.
6. **OLAP Functions**
   OLAP functions returns data from a dimensional object in two-dimension relational format.
7. **Data Cartridge Functions**
   Facilitate Data Cartridge development.

Single-Row Functions

1. Numeric Functions
2. Character Functions Returning Character Values
3. Character Functions Returning Number Values
4. NLS Character Functions
5. Datetime Functions
6. General Comparison Functions
7. Conversion Functions
8. NULL-Related Functions
9. Large Object Functions
10. Collection Functions
11. Hierarchical Functions
12. Data Mining Functions
13. XML Functions
14. Encoding and Decoding Functions
15. Environment and Identifier Functions

Subject of this module

Numeric Functions

1. **ABS**
2. ACOS, ASIN
3. ATAN, ATAN2
4. BITAND
5. **CEIL**
6. COS, COSH
7. EXP
8. **FLOOR**
9. LN, LOG
10. **MOD**
11. NANVL
12. POWER
13. **REMAINDER**
14. **ROUND (number), TRUNC (number)**
15. SIGN
16. SIN, SINH
17. **SQRT**
18. TAN, TANH
19. WIDTH_BUCKET

The most useful functions are marked blue

Character Functions

Character Functions Returning Character Values

1. CHR, NCHR
2. CONCAT
3. INITCAP, LOWER, UPPER
4. NLS_INITCAP, NLS_LOWER, NLS_UPPER
5. LPAD, RPAD
6. TRIM, LTRIM, RTRIM
7. NLSSORT
8. REGEXP_REPLACE, REGEXP_SUBSTR
9. REPLACE
10. SOUNDEX
11. SUBSTR
12. TRANSLATE

Character Functions Returning Number Values

1. ASCII
2. INSTR
3. LENGTH
4. REGEXP_COUNT, REGEXP_INSTR

Datetime Functions

1. ADD_MONTHS
2. CURRENT_DATE, CURRENT_TIMESTAMP
3. DBTIMEZONE
4. EXTRACT (datetime)
5. FROM_TZ
6. LAST_DAY, NEXT_DAY
7. LOCALTIMESTAMP
8. MONTHS_BETWEEN
9. NEW_TIME
10. NUMTODSINTERVAL, NUMTOYMINTERVAL
11. ORA_DST_AFFECTED, ORA_DST_CONVERT, ORA_DST_ERROR
12. ROUND (date), TRUNC (date)
13. SESSIONTIMEZONE
14. SYS_EXTRACT_UTC
15. SYSDATE, SYSTIMESTAMP
16. TO_CHAR (datetime)
17. TO_DSINTERVAL, TO_YMINTERVAL
18. TO_TIMESTAMP, TO_TIMESTAMP_TZ
19. TZ_OFFSET

General Comparison Functions

1. GREATEST
2. LEAST

Conversion Functions

1. ASCIISTR
2. BIN_TO_NUM
3. CAST
4. CHARTOROWID
5. COMPOSE
6. CONVERT
7. DECOMPOSE
8. HEXTORAW
9. NUMTODSINTERVAL
10. NUMTOYMINTERVAL
11. RAWTOHEX
12. RAWTONHEX
13. ROWIDTOCHAR
14. ROWIDTONCHAR
15. SCN_TO_TIMESTAMP
16. TIMESTAMP_TO_SCN
17. TO_BINARY_DOUBLE
18. TO_BINARY_FLOAT
19. TO_BLOB
20. TO_CHAR (character)
21. TO_CHAR (datetime)
22. TO_CHAR (number)
23. TO_CLOB
24. TO_DATE
25. TO_DSINTERVAL
26. TO_LOB
27. TO_MULTI_BYTE
28. TO_NCHAR (character)
29. TO_NCHAR (datetime)
30. TO_NCHAR (number)
31. TO_NCLOB
32. TO_NUMBER
33. TO_SINGLE_BYTE
34. TO_TIMESTAMP
35. TO_TIMESTAMP_TZ
36. TO_YMINTERVAL
37. TRANSLATE ... USING
38. UNISTR

Large Object Functions

1. **BFILENAME**
2. **EMPTY_BLOB**
3. **EMPTY_CLOB**

Hierarchical Functions

1. **SYS_CONNECT_BY_PATH**

Data Mining Functions

1. CLUSTER_ID
2. CLUSTER_PROBABILITY
3. CLUSTER_SET
4. FEATURE_ID
5. FEATURE_SET
6. FEATURE_VALUE
7. PREDICTION
8. PREDICTION_BOUNDS
9. PREDICTION_COST
10. PREDICTION_DETAILS
11. PREDICTION_PROBABILITY
12. PREDICTION_SET

XML Functions

1. APPENDCHILDXML
2. DELETEXML
3. DEPTH
4. EXISTSNODE
5. EXTRACT (XML)
6. EXTRACTVALUE
7. INSERTCHILDXML
8. INSERTCHILDXMLAFTER
9. INSERTCHILDXMLBEFORE
10. INSERTXMLAFTER
11. INSERTXMLBEFORE
12. PATH
13. SYS_DBURIGEN
14. SYS_XMLAGG
15. SYS_XMLGEN
16. UPDATEXML
17. XMLAGG
18. XMLCAST
19. XMLCDATA
20. XMLCOLATTVAL
21. XMLCOMMENT
22. XMLCONCAT
23. XMLDIFF
24. XMLELEMENT
25. XMLEXISTS
26. XMLFOREST
27. XMLISVALID
28. XMLPARSE
29. XMLPATCH
30. XMLPI
31. XMLQUERY
32. XMLROOT
33. XMLSEQUENCE
34. XMLSERIALIZE
35. XMLTABLE
36. XMLTRANSFORM

Encoding and Decoding Functions

1. DECODE
2. DUMP
3. ORA_HASH
4. VSIZE

NULL-Related Functions

1. COALESCE
2. LNNVL
3. NANVL
4. NULLIF
5. NVL
6. NVL2

Environment and Identifier Functions

1. SYS_CONTEXT
2. SYS_GUID
3. SYS_TYPEID
4. UID
5. USER
6. USERENV

Aggregate Functions

Aggregate functions return a single result row based on groups of rows, rather than on single rows.

Aggregate functions can appear in select lists, in ORDER BY and HAVING clauses. They are commonly used with the GROUP BY clause in a SELECT statement, where Oracle Database divides the rows of a queried table or view into groups.

In a query containing a GROUP BY clause, the elements of the select list can be aggregate functions, GROUP BY expressions, constants, or expressions involving one of these. Oracle applies the aggregate functions to each group of rows and returns a single result row for each group.

If you omit the GROUP BY clause, then Oracle applies aggregate functions in the select list to all the rows in the queried table or view.

Use aggregate functions in the HAVING clause to eliminate groups from the output based on the results of the aggregate functions, rather than on the values of the individual rows of the queried table or view.

DISTINCT (UNIQUE) / ALL

Many (but not all) aggregate functions that take a single argument accept these clauses:

- DISTINCT and UNIQUE, which are synonymous, cause an aggregate function to consider only distinct values of the argument expression.
- ALL causes an aggregate function to consider all values, including all duplicates.

- DISTINCT average of 1, 1, 1, and 3 is 2.
- The ALL average is 1.5.
- If you specify neither, then the default is ALL.

All aggregate functions except COUNT(*), GROUPING, and GROUPING_ID ignore nulls. You can use the NVL function in the argument to an aggregate function to substitute a value for a null.

COUNT and REGR_COUNT never return null, but return either a number or zero.

For all the remaining aggregate functions, if the data set contains no rows, or contains only rows with nulls as arguments to the aggregate function, then the function returns null.

Nested Aggregates

You can nest aggregate functions. For example, the following statement calculates the average of the maximum salaries of all the departments in the sample schema HR:

```
SELECT AVG(MAX(salary))
FROM employees
GROUP BY department_id;
```

```
AVG(MAX(SALARY))
----------------
10926.3333
```

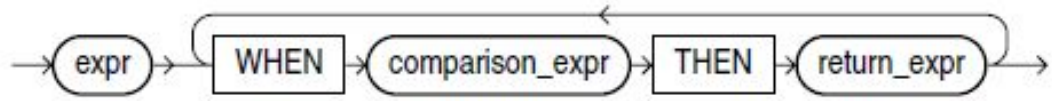
Aggregate Functions

1. AVG
2. COLLECT
3. CORR
4. CORR_*
5. COUNT
6. COVAR_POP
7. COVAR_SAMP
8. CUME_DIST
9. DENSE_RANK
10. FIRST
11. GROUP_ID
12. GROUPING
13. GROUPING_ID
14. LAST
15. LISTAGG
16. MAX
17. MEDIAN
18. MIN
19. PERCENT_RANK
20. PERCENTILE_CONT
19. PERCENTILE_DISC
20. RANK
21. REGR_ (Linear Regression) Functions
22. STATS_BINOMIAL_TEST
23. STATS_CROSSTAB
24. STATS_F_TEST
25. STATS_KS_TEST
26. STATS_MODE
27. STATS_MW_TEST
28. STATS_ONE_WAY_ANOVA
29. STATS_T_TEST_*
30. STATS_WSR_TEST
31. STDDEV
32. STDDEV_POP
33. STDDEV_SAMP
34. SUM
35. SYS_XMLAGG
36. VAR_POP
37. VAR_SAMP
38. VARIANCE
39. XMLAGG

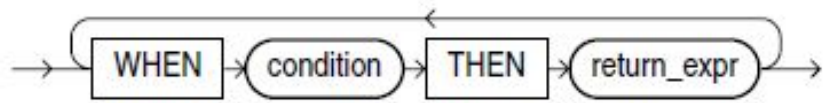
CASE Expressions

CASE expressions let you use IF ... THEN ... ELSE logic in SQL statements without having to invoke procedures.

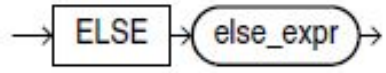
***simple_case_expression*::=**



***searched_case_expression*::=**



***else_clause*::=**



Simple Case (or Case with Selector) Example

```
SELECT first_name, last_name,
  CASE trunc(salary/5000)
    WHEN 0 THEN 'LOW-PAID'
    WHEN 1 THEN 'MID-PAID'
    WHEN 2 THEN 'WELL-PAID'
    ELSE 'EXCELLENT'
  END AS SALARY_CATEGORY
FROM employees;

SELECT first_name, last_name,
  DECODE(trunc(salary/5000),
    0, 'LOW-PAID',
    1, 'MID-PAID',
    2, 'WELL-PAID',
    'EXCELLENT') SALARY_CATEGORY
FROM employees;
```

	FIRST_NAME	LAST_NAME	SALARY_CATEGORY
1	Steven	King	EXCELLENT
2	Neena	Kochhar	EXCELLENT
3	Lex	De Haan	EXCELLENT
4	Alexander	Hunold	MID-PAID
5	Bruce	Ernst	MID-PAID
6	David	Austin	LOW-PAID
7	Valli	Pataballa	LOW-PAID
8	Diana	Lorentz	LOW-PAID
9	Nancy	Greenberg	WELL-PAID
10	Daniel	Faviet	MID-PAID
11	John	Chen	MID-PAID
12	Ismael	Sciarra	MID-PAID
13	Jose Manuel	Urman	MID-PAID
14	Luis	Popp	MID-PAID
15	Den	Raphaely	WELL-PAID
16	Alexander	Khoo	LOW-PAID

Searched Case Example

```
SELECT first_name, last_name,
  CASE
    WHEN salary < 5000 THEN 'LOW-PAID'
    WHEN salary >= 5000 AND salary < 10000 THEN 'MID-PAID'
    WHEN salary >= 10000 AND salary < 15000 THEN 'WELL-PAID'
    ELSE 'EXCELLENT'
  END AS SALARY_CATEGORY
FROM employees;
```

```
SELECT first_name, last_name,
  CASE
    WHEN salary < 5000
      THEN 'LOW-PAID'
    WHEN salary < 10000
      THEN 'MID-PAID'
    WHEN salary < 15000
      THEN 'WELL-PAID'
    ELSE 'EXCELLENT'
END AS SALARY_CATEGORY
FROM employees;
```

	FIRST_NAME	LAST_NAME	SALARY_CATEGORY
1	Steven	King	EXCELLENT
2	Neena	Kochhar	EXCELLENT
3	Lex	De Haan	EXCELLENT
4	Alexander	Hunold	MID-PAID
5	Bruce	Ernst	MID-PAID
6	David	Austin	LOW-PAID
7	Valli	Pataballa	LOW-PAID
8	Diana	Lorentz	LOW-PAID
9	Nancy	Greenberg	WELL-PAID
10	Daniel	Faviet	MID-PAID
11	John	Chen	MID-PAID
12	Ismael	Sciarra	MID-PAID
13	Jose Manuel	Urman	MID-PAID
14	Luis	Popp	MID-PAID
15	Den	Raphaely	WELL-PAID
16	Alexander	Khoo	LOW-PAID

SUBQUERIES

Define Subqueries

Subquery is a SELECT statement that is nested within another SQL statement.

SQL statements those accept subqueries:

- **DML: SELECT, INSERT, UPDATE, DELETE, MERGE**
- **DDL: CREATE TABLE and CREATE VIEW**

A SQL statement that includes a subquery as part of its code is considered the parent (or outer) to the subquery (or inner query).
A parent SQL statement may include one or more subqueries in its syntax.
Subqueries may have their own subqueries.

```
SELECT FIRST_NAME,
  LAST_NAME,
  SALARY
FROM EMPLOYEES
WHERE SALARY >= (SELECT MAX(SALARY)*0.7 FROM EMPLOYEES)
ORDER BY SALARY DESC;
```

FIRST_NAME	LAST_NAME	SALARY
Steven	King	24000
Lex	De Haan	17000
Neena	Kochhar	17000

Compare two queries

```
SELECT FIRST_NAME,
   LAST_NAME,
   SALARY
FROM EMPLOYEES
WHERE SALARY = (SELECT MAX(SALARY) FROM EMPLOYEES)
ORDER BY SALARY DESC;
```

```
SELECT *
FROM (
   SELECT FIRST_NAME,
     LAST_NAME,
     SALARY
   FROM EMPLOYEES
   ORDER BY SALARY DESC
)
WHERE ROWNUM=1;
```

- **Both return identical dataset**
- **The second causes a mistake when more than one person has the greatest salary**
- **Both use subqueries, but the nature of subqueries is different:**
  - In the first – **subquery acts like scalar** (just number – maximum salary value)
  - In the second – **subquery acts like dataset** (row source).

```
FIRST_NAME       LAST_NAME      SALARY
---------------  -----------  --------
Steven           King            24000
```

Subqueries Classification

- **Single-row subqueries**
  Return a single row in its result
- **Multiple-row subqueries**
  Return zero, one, or more rows
- **Multiple-column subqueries**
  Return more than one column in its result
- **Scalar subqueries**
  A single-row subquery consists of only one column
- **Correlated subqueries**
  Reference column(columns) from the parent query(queries)

```
SELECT FIRST_NAME,
  LAST_NAME,
  SALARY
FROM EMPLOYEES
WHERE SALARY = (
  SELECT MAX(SALARY)
  FROM EMPLOYEES
)
ORDER BY SALARY DESC;
```

```
SELECT *
FROM (
  SELECT FIRST_NAME,
    LAST_NAME,
    SALARY
  FROM EMPLOYEES
  ORDER BY SALARY DESC
)
WHERE ROWNUM=1;
```

Single-row subquery

```
SELECT EMP.FIRST_NAME,
  EMP.LAST_NAME,
  EMP.JOB_ID
FROM EMPLOYEES EMP
WHERE
  (SUBSTR(EMP.LAST_NAME,1,1), SUBSTR(EMP.LAST_NAME,2,1)) = (
    SELECT CHR(ROUND(AVG(ASCII(SUBSTR(LAST_NAME,1,1)))),
      CHR(MEDIAN(ASCII(SUBSTR(LAST_NAME,2,1))))
    FROM EMPLOYEES E
  );
```

```
FIRST_NAME   LAST_NAME   JOB_ID
------------ ----------- ----------
Alexander    Khoo        PU_CLERK
```

```
SELECT EMP.FIRST_NAME,
  EMP.LAST_NAME,
  EMP.JOB_ID
FROM EMPLOYEES EMP
WHERE
  SUBSTR(EMP.LAST_NAME,1,1) || SUBSTR(EMP.LAST_NAME,2,1) = (
    SELECT CHR(ROUND(AVG(ASCII(SUBSTR(LAST_NAME,1,1))))) ||
      CHR(MEDIAN(ASCII(SUBSTR(LAST_NAME,2,1))))
    FROM EMPLOYEES E
  );
```

Multiple-row subqueries

```
SELECT MANAGERS.EMPLOYEE_ID, MANAGERS.FIRST_NAME,
  MANAGERS.LAST_NAME, MANAGERS.SALARY
FROM (
  SELECT E.EMPLOYEE_ID, E.FIRST_NAME, E.LAST_NAME, E.SALARY
  FROM DEPARTMENTS D
    JOIN EMPLOYEES E ON (D.MANAGER_ID = E.EMPLOYEE_ID)
) HEADS_OF_DEPTS JOIN (
  SELECT DISTINCT MGR.EMPLOYEE_ID, MGR.FIRST_NAME,
    MGR.LAST_NAME, MGR.SALARY
  FROM EMPLOYEES E
    JOIN EMPLOYEES MGR ON (E.MANAGER_ID = MGR.EMPLOYEE_ID)
) MANAGERS
ON (HEADS_OF_DEPTS.EMPLOYEE_ID = MANAGERS.EMPLOYEE_ID);
SELECT E.EMPLOYEE_ID, E.FIRST_NAME, E.LAST_NAME, E.SALARY
FROM DEPARTMENTS D
  JOIN EMPLOYEES E ON (D.MANAGER_ID = E.EMPLOYEE_ID)
INTERSECT
SELECT DISTINCT MGR.EMPLOYEE_ID, MGR.FIRST_NAME,
  MGR.LAST_NAME, MGR.SALARY
FROM EMPLOYEES E
  JOIN EMPLOYEES MGR ON (E.MANAGER_ID = MGR.EMPLOYEE_ID);
```

JOIN-based equivalent

```
SELECT DISTINCT
  MGR.EMPLOYEE_ID, MGR.FIRST_NAME, MGR.LAST_NAME, MGR.SALARY
FROM (
    DEPARTMENTS D
    JOIN EMPLOYEES DMGR ON (D.MANAGER_ID = DMGR.EMPLOYEE_ID))
  JOIN (
    EMPLOYEES E
    JOIN EMPLOYEES MGR ON (E.MANAGER_ID = MGR.EMPLOYEE_ID))
  ON DMGR.EMPLOYEE_ID = MGR.EMPLOYEE_ID
;
```

MPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
145	John	Russell	14000
201	Michael	Hartstein	13000
108	Nancy	Greenberg	12008
205	Shelley	Higgins	12008
114	Den	Raphaely	11000
103	Alexander	Hunold	9000
121	Adam	Fripp	8200

Comparison conditions for multiple-row subqueries

1. **IN. Compares a subject value to a set of values. Returns TRUE if the subject value equals any of the values in the set. Returns FALSE if the subquery returns no rows.**
2. **NOT IN. NOT used with IN to reverse the result. Returns TRUE if the subquery returns no rows.**
3. **EXISTS. An EXISTS condition tests for existence of rows in a subquery. Returns TRUE if a subquery returns at least one row.**

Employees with maximum salaries by jobs (ANY, IN)

```
-- ORA-01427: single-row subquery returns more than one row
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMPLOYEES
WHERE
  SALARY = (SELECT MAX(SALARY) FROM EMPLOYEES GROUP BY JOB_ID)
ORDER BY SALARY DESC;


-- Invalid logic
SELECT FIRST_NAME, LAST_NAME, SALARY
FROM EMPLOYEES
WHERE
  SALARY IN (SELECT MAX(SALARY) FROM EMPLOYEES GROUP BY JOB_ID)
ORDER BY SALARY DESC;
```

Employees with maximum salaries by jobs (Correlated subquery)

```
SELECT EMP.FIRST_NAME,
  EMP.LAST_NAME,
  JOB_ID,
  EMP.SALARY
FROM EMPLOYEES EMP
WHERE SALARY = (
  SELECT MAX(E.SALARY)
  FROM EMPLOYEES E
  WHERE E.JOB_ID = EMP.JOB_ID
)
ORDER BY EMP.SALARY DESC;
```

FIRST_NAME	LAST_NAME	JOB_ID	SALARY
Steven	King	AD_PRES	24000
Neena	Kochhar	AD_VP	17000
Lex	De Haan	AD_VP	17000
John	Russell	SA_MAN	14000
Michael	Hartstein	MK_MAN	13000
Nancy	Greenberg	FI_MGR	12008
Shelley	Higgins	AC_MGR	12008
Lisa	Ozer	SA_REP	11500
Den	Raphaely	PU_MAN	11000
Hermann	Baer	PR_REP	10000
Alexander	Hunold	IT_PROG	9000
Daniel	Faviet	FI_ACCOUNT	9000
William	Gietz	AC_ACCOUNT	8300
Adam	Fripp	ST_MAN	8200
Susan	Mavris	HR_REP	6500
Pat	Fay	MK_REP	6000
Jennifer	Whalen	AD_ASST	4400
Nandita	Sarchand	SH_CLERK	4200
Renske	Ladwig	ST_CLERK	3600
Alexander	Khoo	PU_CLERK	3100

Employees with maximum salaries by jobs (EXIST, ALL)

```
SELECT EMP.FIRST_NAME,
  EMP.LAST_NAME,
  EMP.JOB_ID,
  EMP.SALARY
FROM EMPLOYEES EMP
WHERE NOT EXISTS (
 SELECT *
 FROM EMPLOYEES E
 WHERE E.SALARY > EMP.SALARY
   AND E.JOB_ID = EMP.JOB_ID
)
ORDER BY EMP.SALARY DESC;
```

FIRST_NAME	LAST_NAME	JOB_ID	SALARY
Steven	King	AD_PRES	24000
Neena	Kochhar	AD_VP	17000
Lex	De Haan	AD_VP	17000
John	Russell	SA_MAN	14000
Michael	Hartstein	MK_MAN	13000
Nancy	Greenberg	FI_MGR	12008
Shelley	Higgins	AC_MGR	12008
Lisa	Ozer	SA_REP	11500
Den	Raphaely	PU_MAN	11000
Hermann	Baer	PR_REP	10000
Alexander	Hunold	IT_PROG	9000
Daniel	Faviet	FI_ACCOUNT	9000
William	Gietz	AC_ACCOUNT	8300
Adam	Fripp	ST_MAN	8200
Susan	Mavris	HR_REP	6500
Pat	Fay	MK_REP	6000
Jennifer	Whalen	AD_ASST	4400
Nandita	Sarchand	SH_CLERK	4200
Renske	Ladwig	ST_CLERK	3600
Alexander	Khoo	PU_CLERK	3100

MTN.BI.02

ORACLE SQL FOUNDATION

Questions & Answers

Author: **Aliaksandr Chaika**
Senior Software Engineer
Certified Oracle Database SQL Expert
Aliaksandr_Chaika@epam.com