ORACLE®
JUNIOR PROGRAM

Oracle Database
SQL Basics

Kerepes Tamás, Webváltó Kft.
tamas.kerepes@webvalto.hu

2015. február 26.

SQL – a history in brief

- The relational database stores data in tables: rows and columns. The idea was suggested by Codd in 1970. Four years later, in 1974 the SQL language was proposed for managing data stored in relational database management systems.
- This form of storing data needs a special language for:
  - Retrieving data
  - Manipulating data (inserting, deleting and updating)
  - Defining and modifying table structures (creating tables, dropping and modifying them). We need commands to define other structures also: indexes, views, synonyms, and so on.
- SQL – Structured Query Language is used nowadays by the vast majority of relational database management systems as this language.

SQL Basics

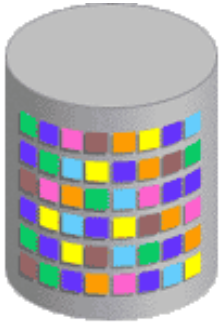
- **Operators**
  Relational: <,>,=,!=,<>
  SQL: IN, LIKE, BETWEEN, IS NULL (NOT can be used)
- **Functions**
  Numeric: ROUND,TRUNC, FLOOR,CEIL,SQRT,SIN,etc.
  Character: SUBSTR, INSTR,REPLACE, UPPER, LOWER
  Date: SYSDATE,SYSTIMESTAMP, ADD_MONTHS
  Converson:TO_CHAR,TO_DATE, TO_NUMBER,TO_LOB
  Aggregate: MIN, MAX, SUM, AVG, COUNT, LISTAGG,COV
  Analytic: LAG, LEAD, WITH_BUCKET,NTILE,
  Other: XML, REGEXP, CHR,ASCII, DECODE, DUMP
- **Statements**
  DML: SELECT, INSERT, UPDATE,DELETE
  DDL: CREATE, DROP, ALTER, RENAME, TRUNCATE
  DCL: GRANT, REVOKE

CREATE TABLE Statement

- You must have:
  - CREATE TABLE system privilege
  - A storage area

```
CREATE TABLE [schema.]table
    (column datatype [DEFAULT expr][, ...]);
```

- You specify:
  - Table name
  - Column name, column data type, and column size



Creating Tables

- Create the table.

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
Table created.
```

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
  - Merge rows into a table from source tables
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:

```
INSERT INTO table [(column [, column...])]
VALUES      (value [, value...]);
```

- With this syntax, only one row can be inserted at a time.
- Insert values into the table in the following order of the columns:

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

Copying Rows from Another Table

- Write your `INSERT` statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows created.
```

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause with the number of columns in the subquery.

UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement:

```
UPDATE table
SET    column = value [, column = value, ...]
[WHERE condition];
```

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- Update more than one row at a time (if required).

```
UPDATE     copy_emp
SET        department_id = 110;
22 rows updated.
```

Updating Two Columns with a Subquery

Update employee 206's job id and salary to match that of employee 205.

```
UPDATE   employees
SET      (job_id, salary) =
         (SELECT  job_id, salary
                  FROM    employees
                  WHERE   employee_id = 205)
WHERE    employee_id    =  206;
1 row updated.
```

DELETE Statement

You can remove existing rows from a table by using the DELETE statement:

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

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

```
DELETE FROM employees
WHERE  department_id =
(SELECT department_id  FROM   departments
 WHERE  department_name  LIKE '%Public%');
1 row deleted.
```

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- It is a data definition language (DDL) statement rather than a DML statement; cannot be easily undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

Selecting All and specific Columns

```
SELECT * FROM   departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

```
SELECT department_id, location_id
FROM departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null. Use the NVL function!

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
Zlotkey	25200
Abel	39600
Taylor	20640
...	
Gietz	

20 rows selected.

```
SELECT last_name, salary,commission_pct,
12*salary*(1+NVL(commission_pct,0))
FROM   employees;
```

Limiting the Rows That Are Selected

Restrict the rows that are returned by using the `WHERE` clause:

```
SELECT employee_id, last_name,
       job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

SQL Row Limiting Clause in Oracle 12c

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary DESC
FETCH first 5 ROWS ONLY ;
```

Using Comparison operators

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000 ;
```

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500;
```

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

```
SELECT * FROM   employees
WHERE  hire_date >= '31-dec-98' ;
```

```
SELECT * FROM   employees
WHERE  hire_date >= DATE '1998-12-31' ;
```

Using the IN, LIKE and IS NULL operator

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201) ;
```

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL ;
```

Using logical operators

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%' ;
```

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%' ;
```

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

Using the ORDER BY Clause

- Sort the retrieved rows with the ORDER BY clause:
  - ASC: ascending order, default
  - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement:

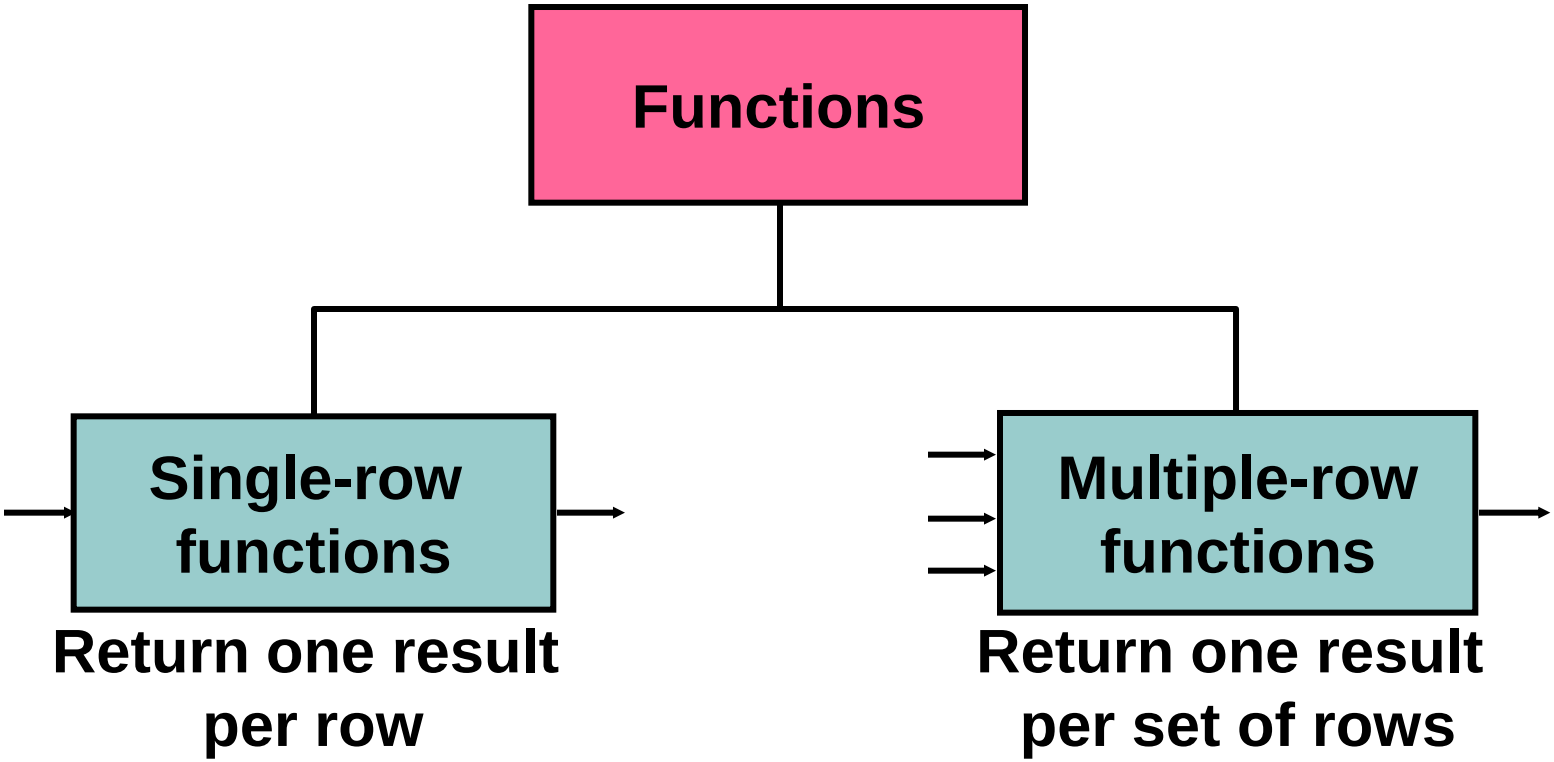
```
SELECT   last_name, job_id,
         department_id, hire_date
FROM     employees
ORDER BY hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

20 rows selected.

Two Types of SQL Functions



Case-Manipulation Functions

These functions convert case for character strings:

Function	Result
`LOWER('SQL Course')`	`sql course`
`UPPER('SQL Course')`	`SQL COURSE`
`INITCAP('SQL Course')`	`Sql Course`

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
`CONCAT('Hello', 'World')`	`HelloWorld`
`SUBSTR('HelloWorld',1,5)`	`Hello`
`LENGTH('HelloWorld')`	`10`
`INSTR('HelloWorld', 'W')`	`6`
`LPAD(salary,10,'*')`	`*****24000`
`RPAD(salary, 10, '*')`	`24000*****`
`REPLACE ('JACK and JUE','J','BL')`	`BLACK and BLUE`
`TRIM('H' FROM 'HelloWorld')`	`elloWorld`

Number Functions

Function	Result
POWER(10,0.3010)	2
SQRT(121)	11
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
CEIL, FLOOR	
POWER(10,2)	100
MOD(1600, 300)	100
SIN,COS,TAN	
LN,SINH,COSH	

```
SELECT SIN(1.57),COS(0),TAN(3.14/4),
POWER(10,0.301),CEIL(1.1),FLOOR(1.9), LN(2.718282)
FROM dual;
```

Date Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Using the TO_CHAR Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

...

20 rows selected.

Using the CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
SELECT last_name, job_id, salary,
  CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
              WHEN 'ST_CLERK' THEN  1.15*salary
              WHEN 'SA_REP'   THEN  1.20*salary
              ELSE  salary     END "REVISED_SALARY"
FROM   employees;
```

```
SELECT last_name,salary,
CASE WHEN salary<5000 THEN 'Low'
     WHEN salary<10000 THEN 'Medium'
     WHEN salary<20000 THEN 'Good'
     ELSE 'Excellent' END qualified_salary
FROM employees;
```

What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

...

20 rows selected.

Maximum salary in the EMPLOYEES table

MAX(SALARY)
24000

Using the AVG, SUM, MIN and MAX Functions

You can use the AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

Creating Groups of Data

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

...

20 rows selected.

4400

9500

3500

6400

10033

Average salary in the EMPLOYEES table for each department

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Using the GROUP BY Clause

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

```
SELECT   department_id dept_id,
         job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;
```

Using the HAVING clause and inline view

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

```
SELECT T.qualified_salary, COUNT(*), SUM(SALARY)
FROM  (SELECT last_name, salary,
         CASE WHEN salary<5000  THEN 'Low'
              WHEN salary<10000 THEN 'Medium'
              WHEN salary<20000 THEN 'Good'
              ELSE 'Excellent'   END qualified_salary
       FROM   employees) T
GROUP BY T.qualified_salary;
```

Displaying Data from Multiple Tables

Joining Column Names

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

...

Foreign key **Primary key**

Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name,
       e.department_id, d.department_id,
       d.location_id
FROM   employees e JOIN departments d
ON     e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

Self-Joins Using the ON Clause

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     worker.manager_id = manager.employee_id;
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King

Retrieving Records with Non-Equijoins

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LEFT,RIGHT and FULL OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

Using a subquery

```
SELECT last_name
FROM   employees
WHERE  salary > (SELECT salary FROM   employees
                 WHERE  last_name = 'Abel');
```

11000

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary = (SELECT MIN(salary)FROM employees);
```

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

Example: IN versus Join

```
SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN
       (SELECT department_id
        FROM departments
        WHERE  location_id = 1700);

-- Versus (!?)

SELECT last_name, e.department_id, job_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    location_id = 1700;

Set serveroutput on
Exec sqlid('location_id = 1700')
```

Examine the execution plans

```
------------------------------------
SELECT last_name, e.department_id, job_id  FROM   employees
e,departments d  WHERE  e.department_id =d.department_id  And
location_id = 1700

Plan hash value: 4017800164

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                   |       |       |     6 (100)|          |
|*  1 |  HASH JOIN                               |                   |   106 |  2862 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| DEPARTMENTS       |    21 |   147 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                      | DEPT_LOCATION_IX  |    21 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS FULL                      | EMPLOYEES         |   107 |  2140 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

```
SELECT last_name, department_id, job_id  FROM   employees  WHERE
department_id IN  (SELECT department_id FROM departments  WHERE
location_id = 1700)

Plan hash value: 4017800164

-----------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                   |       |       |     6 (100)|          |
|*  1 |  HASH JOIN                               |                   |   106 |  2862 |     6   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| DEPARTMENTS       |    21 |   147 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                      | DEPT_LOCATION_IX  |    21 |       |     1   (0)| 00:00:01 |
|   4 |   TABLE ACCESS FULL                      | EMPLOYEES         |   107 |  2140 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```