Introduction to SQL

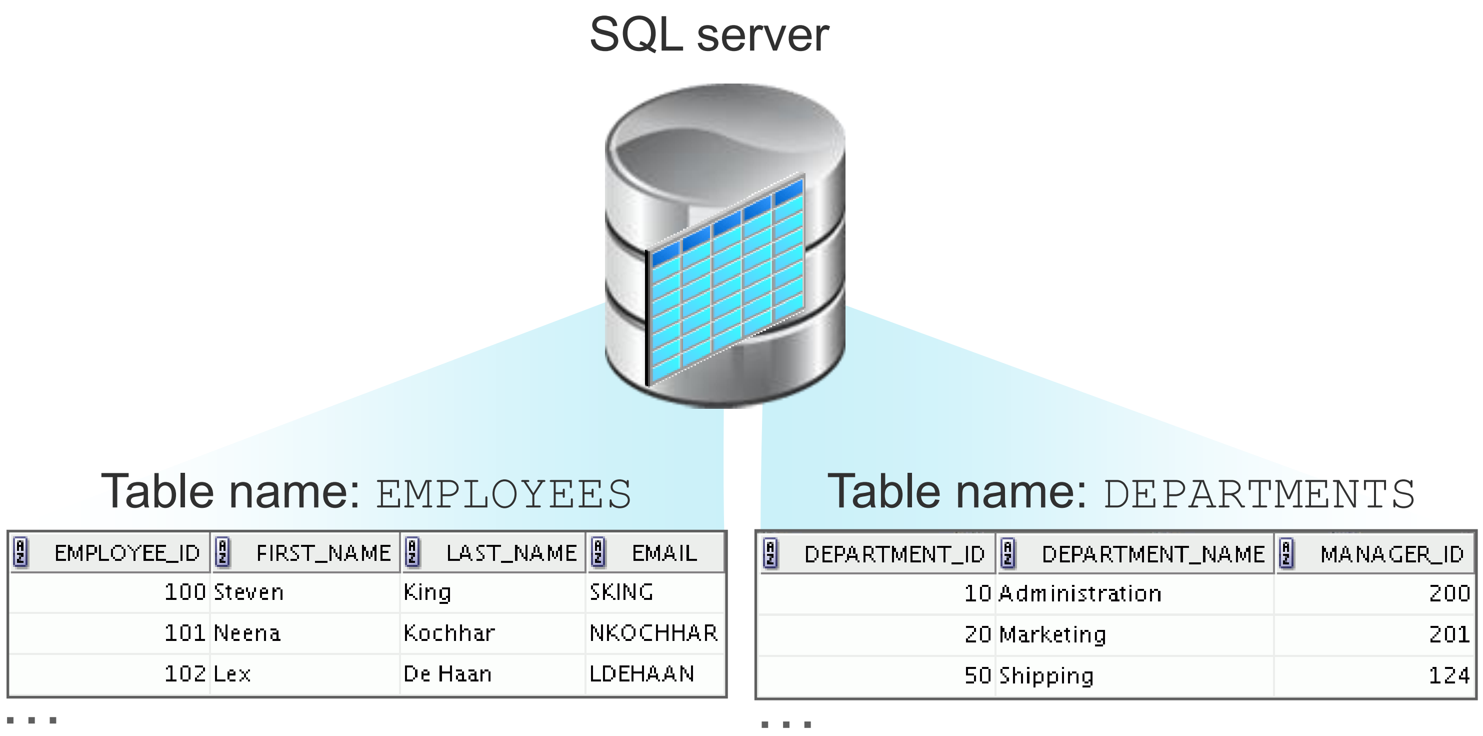
Burak UYSAL

burakuysal@oratekbilisim.com

0532 525 45 36

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables controlled by the server.



SQL server

Table name: `EMPLOYEES`

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
100	Steven	King	SKING
101	Neena	Kochhar	NKOCHHAR
102	Lex	De Haan	LDEHAAN

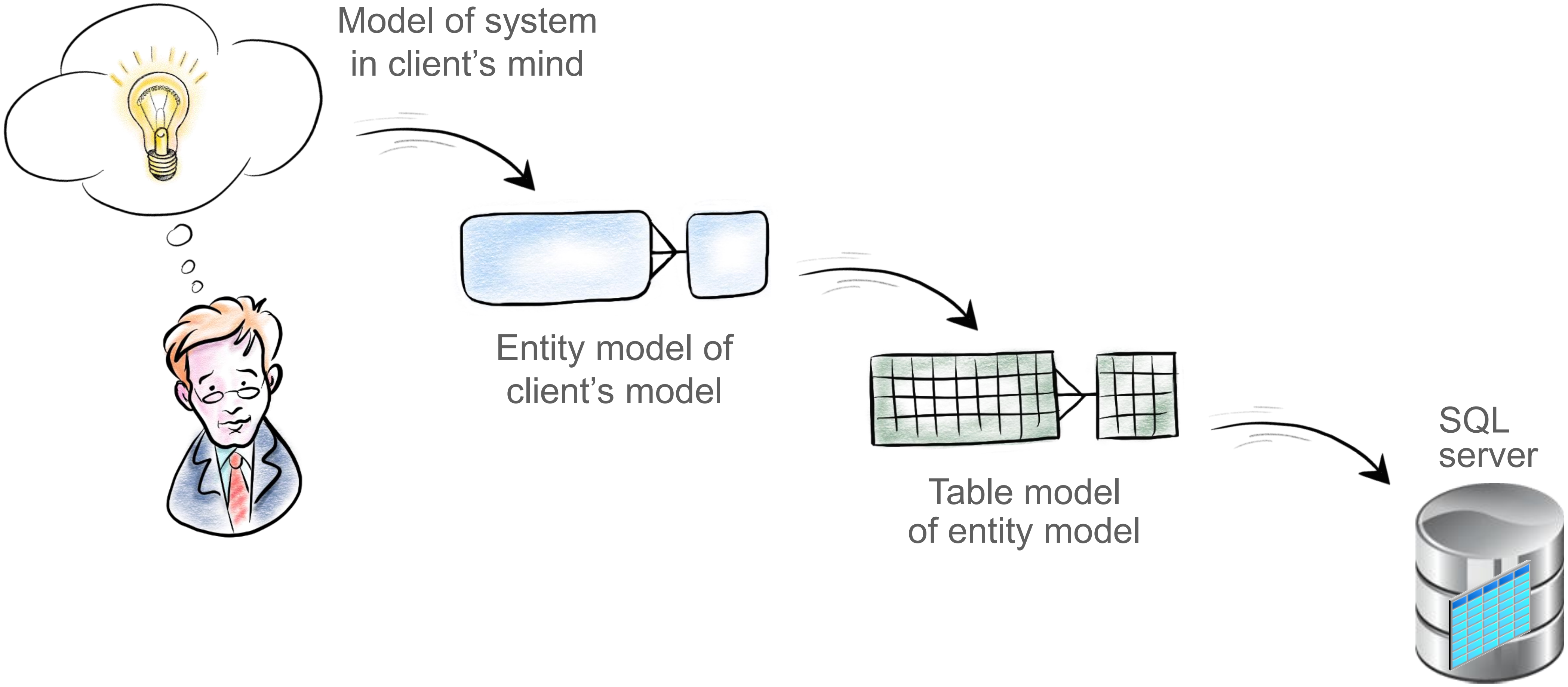
. . .

Table name: `DEPARTMENTS`

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID
10	Administration	200
20	Marketing	201
50	Shipping	124

. . .

Data Models



Model of system in client's mind

Entity model of client's model

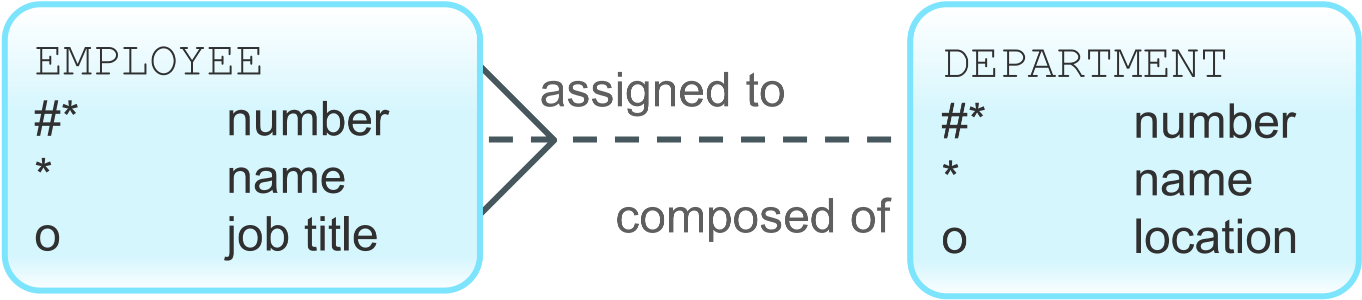
Table model of entity model

SQL server

Tables on disk

Entity Relationship Model

- Create an entity relationship diagram from business specifications or narratives:



EMPLOYEE
#* number
* name
o job title

assigned to

composed of

DEPARTMENT
#* number
* name
o location

- Scenario:
  - “. . . Assign one or more employees to a department. . .”
  - “. . . Some departments do not yet have assigned employees. . .”

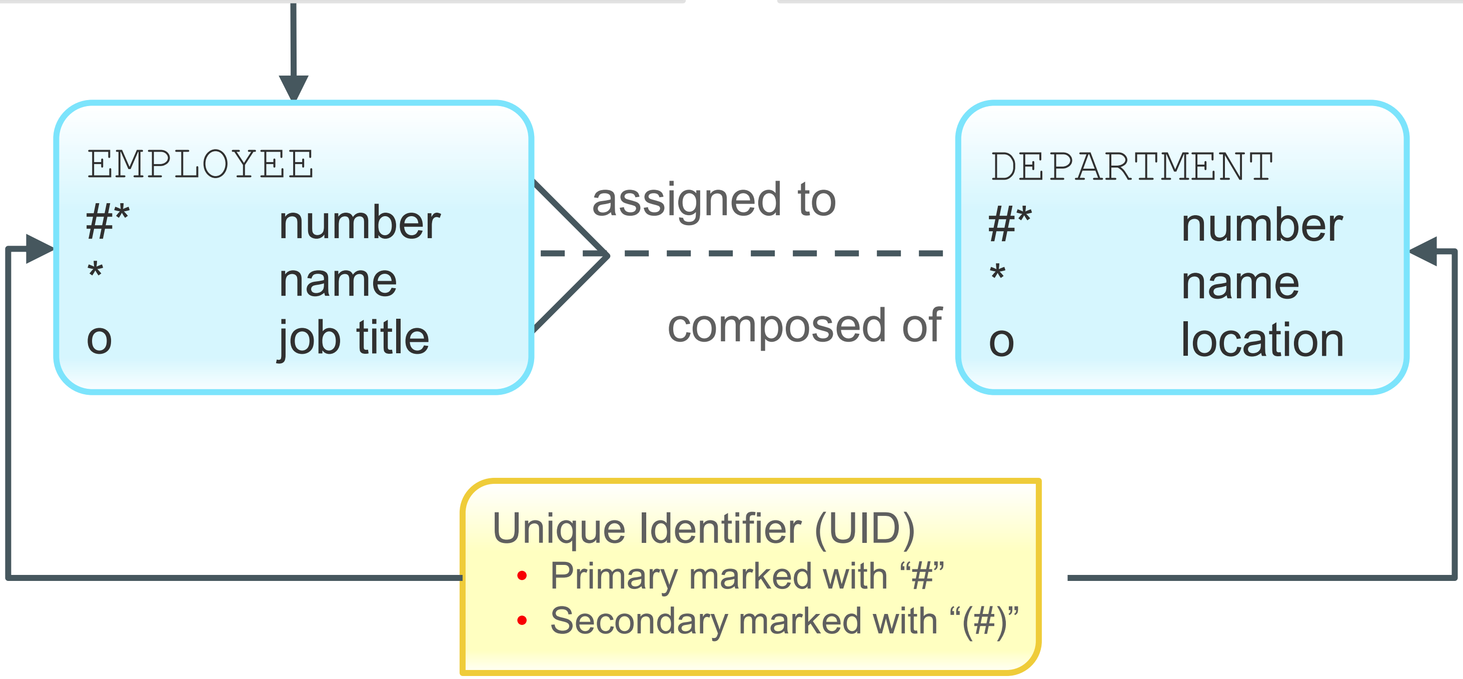
Entity Relationship Modeling Conventions

Entity

- Singular, unique name
- Uppercase
- Soft box
- Synonym in parentheses

Attribute

- Singular name
- Lowercase
- Mandatory marked with ‘*’
- Optional marked with ‘o’



EMPLOYEE
#* number
* name
o job title

assigned to

composed of

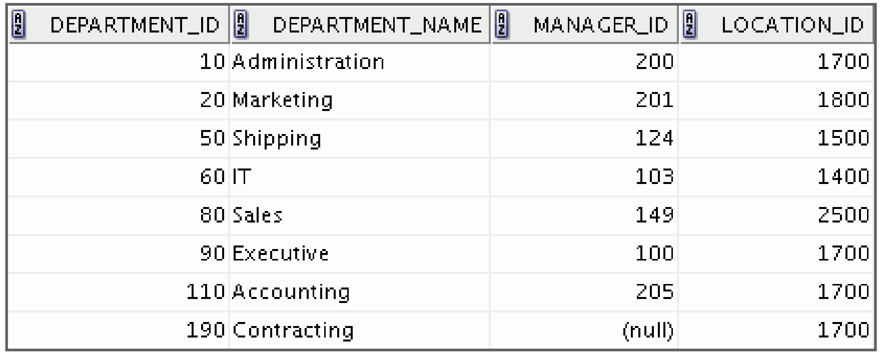
DEPARTMENT
#* number
* name
o location

Unique Identifier (UID)
- Primary marked with “#”
- Secondary marked with “(#)”

Relating Multiple Tables

- Each row of data in a table can be uniquely identified by a primary key.
- You can logically relate data from multiple tables using foreign keys.

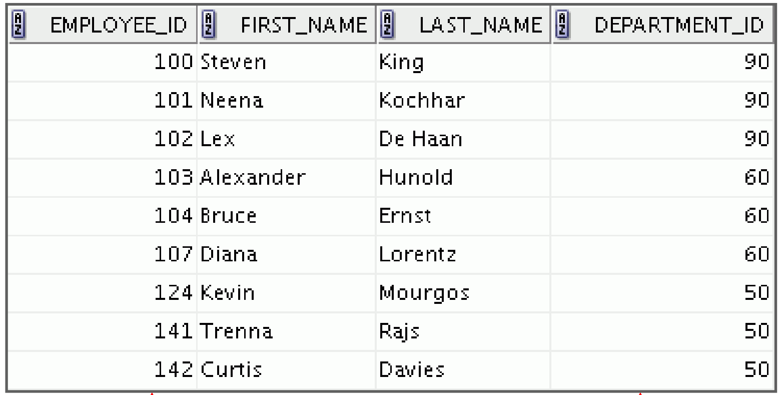
Table name: `DEPARTMENTS`



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	(null)	1700

Primary key

Table name: `EMPLOYEES`

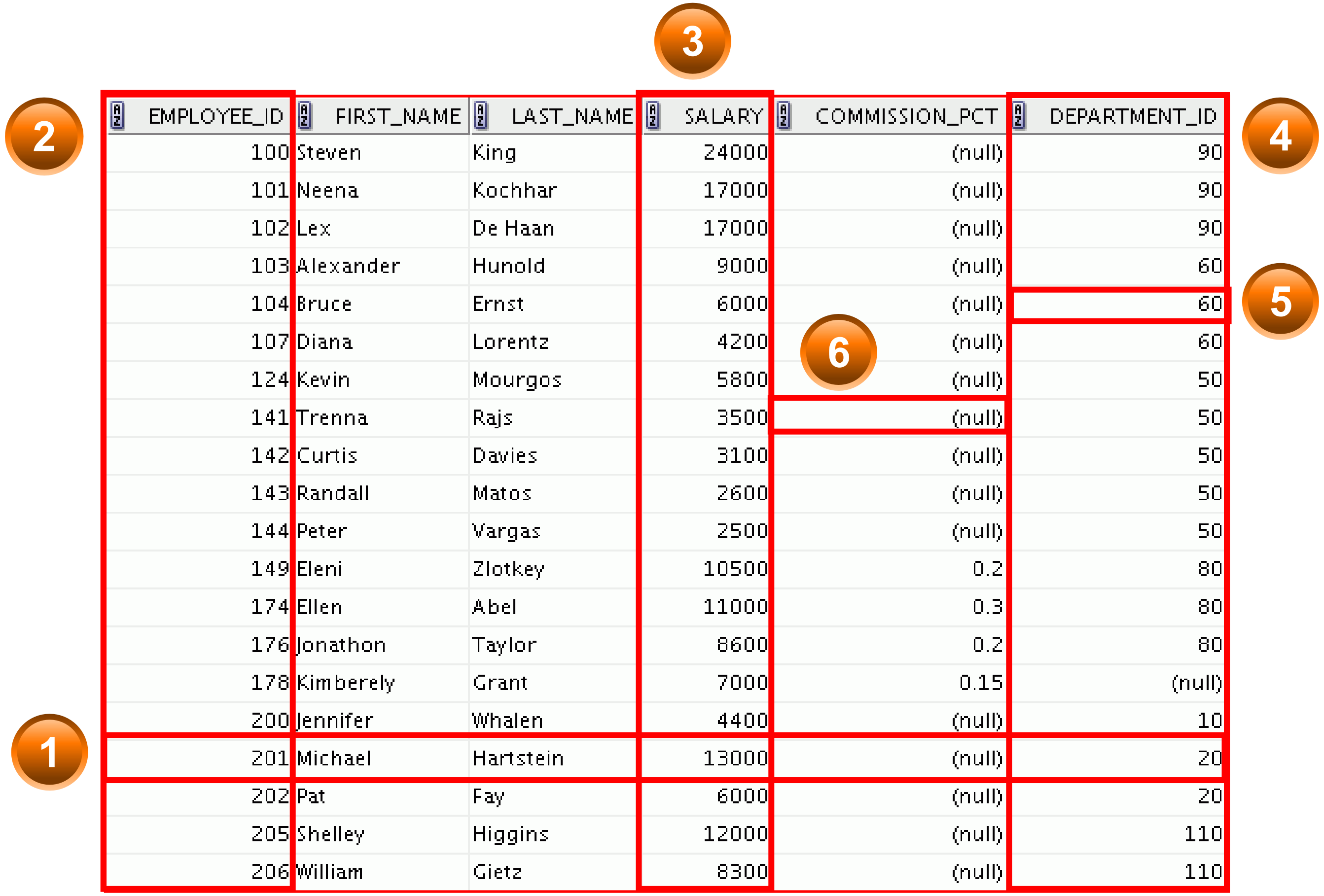


EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60
124	Kevin	Mourgos	50
141	Trenna	Rajs	50
142	Curtis	Davies	50

Primary key

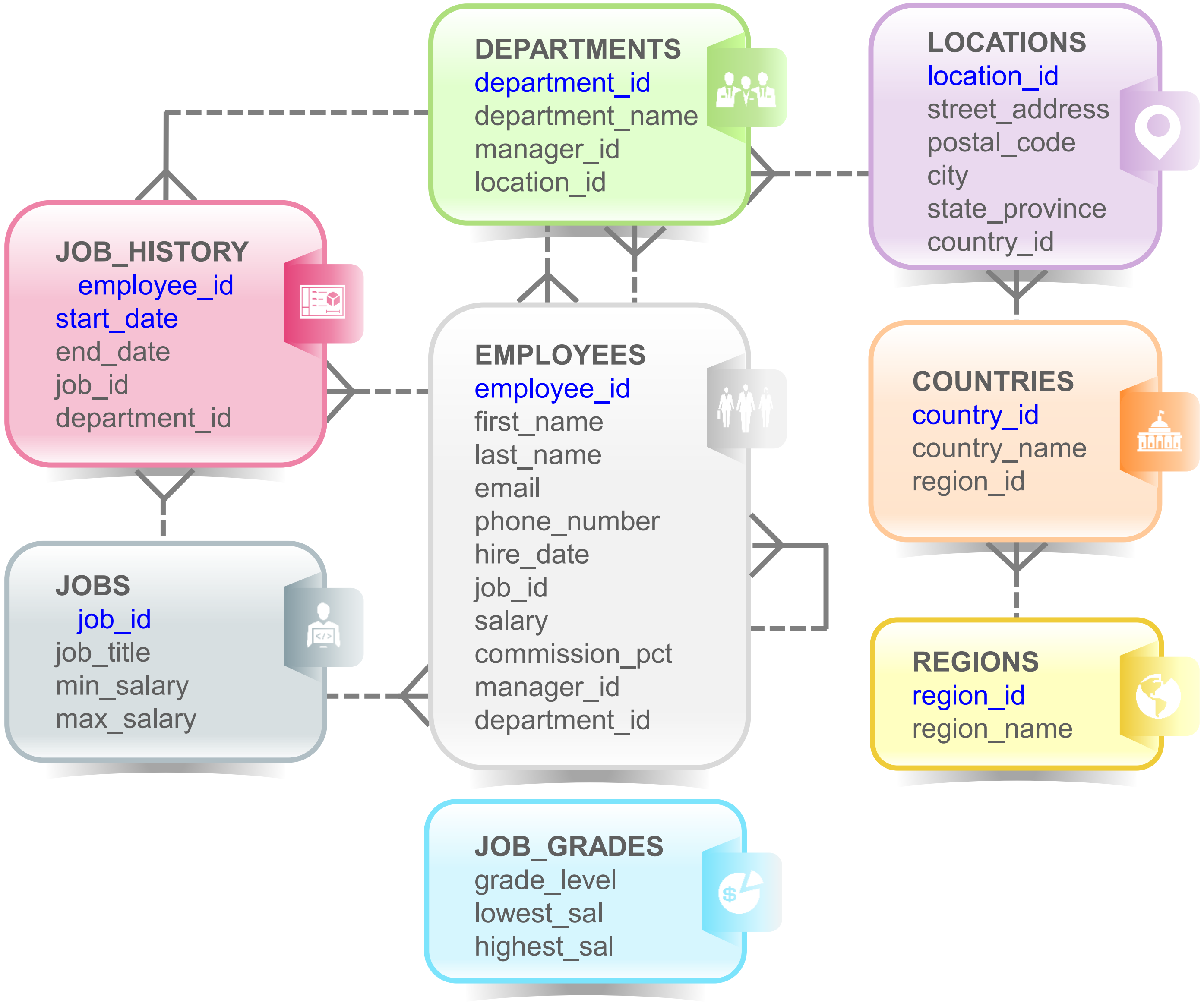
Foreign key

Relational Database Terminology



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	Trenna	Rajs	3500	(null)	50
142	Curtis	Davies	3100	(null)	50
143	Randall	Matos	2600	(null)	50
144	Peter	Vargas	2500	(null)	50
149	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	Jonathon	Taylor	8600	0.2	80
178	Kimberely	Grant	7000	0.15	(null)
200	Jennifer	Whalen	4400	(null)	10
201	Michael	Hartstein	13000	(null)	20
202	Pat	Fay	6000	(null)	20
205	Shelley	Higgins	12000	(null)	110
206	William	Gietz	8300	(null)	110

Tables Used in This Course



DEPARTMENTS
department_id
department_name
manager_id
location_id

LOCATIONS
location_id
street_address
postal_code
city
state_province
country_id

JOB_HISTORY
employee_id
start_date
end_date
job_id
department_id

EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id

COUNTRIES
country_id
country_name
region_id

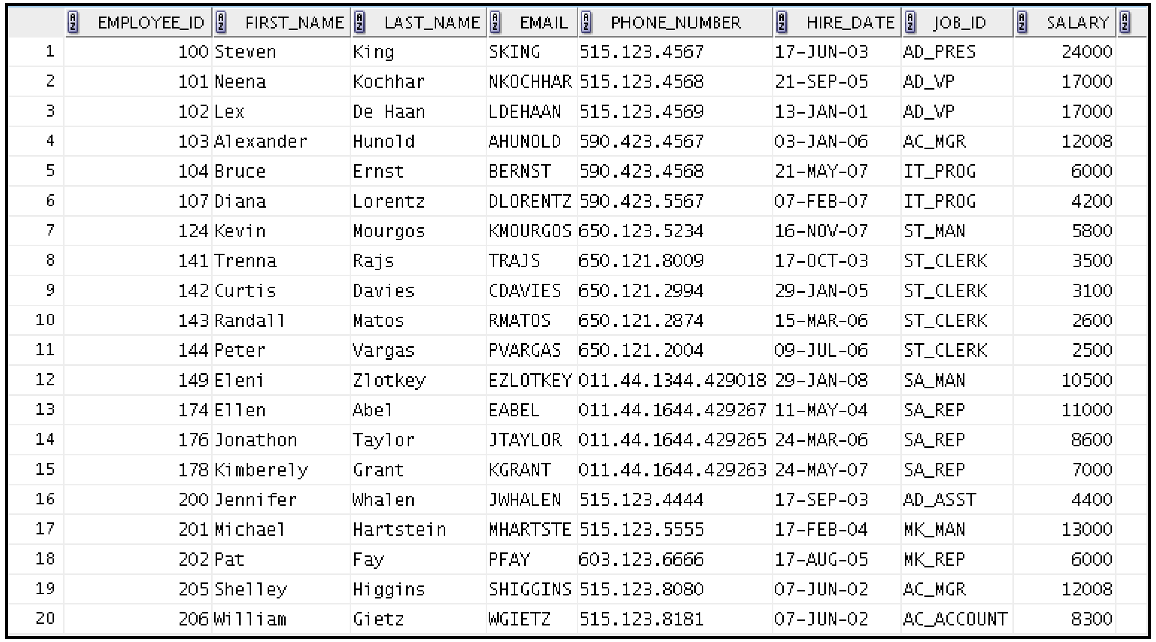
JOBS
job_id
job_title
min_salary
max_salary

REGIONS
region_id
region_name

JOB_GRADES
grade_level
lowest_sal
highest_sal

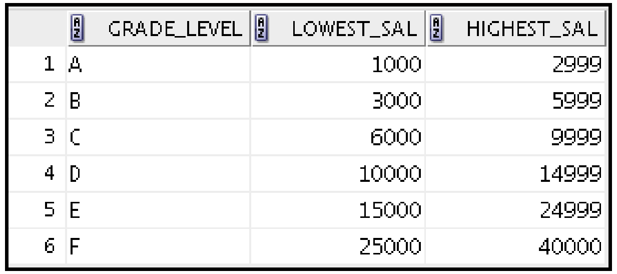
Tables Used in the Course

EMPLOYEES



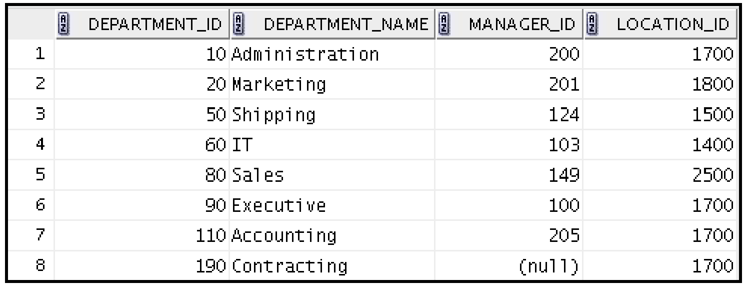
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	AC_MGR	12008
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-03	ST_CLERK	3500
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-05	ST_CLERK	3100
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-06	ST_CLERK	2600
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-06	ST_CLERK	2500
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-08	SA_MAN	10500
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-04	SA_REP	11000
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-06	SA_REP	8600
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-07	SA_REP	7000
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300

JOB_GRADES



	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

DEPARTMENTS

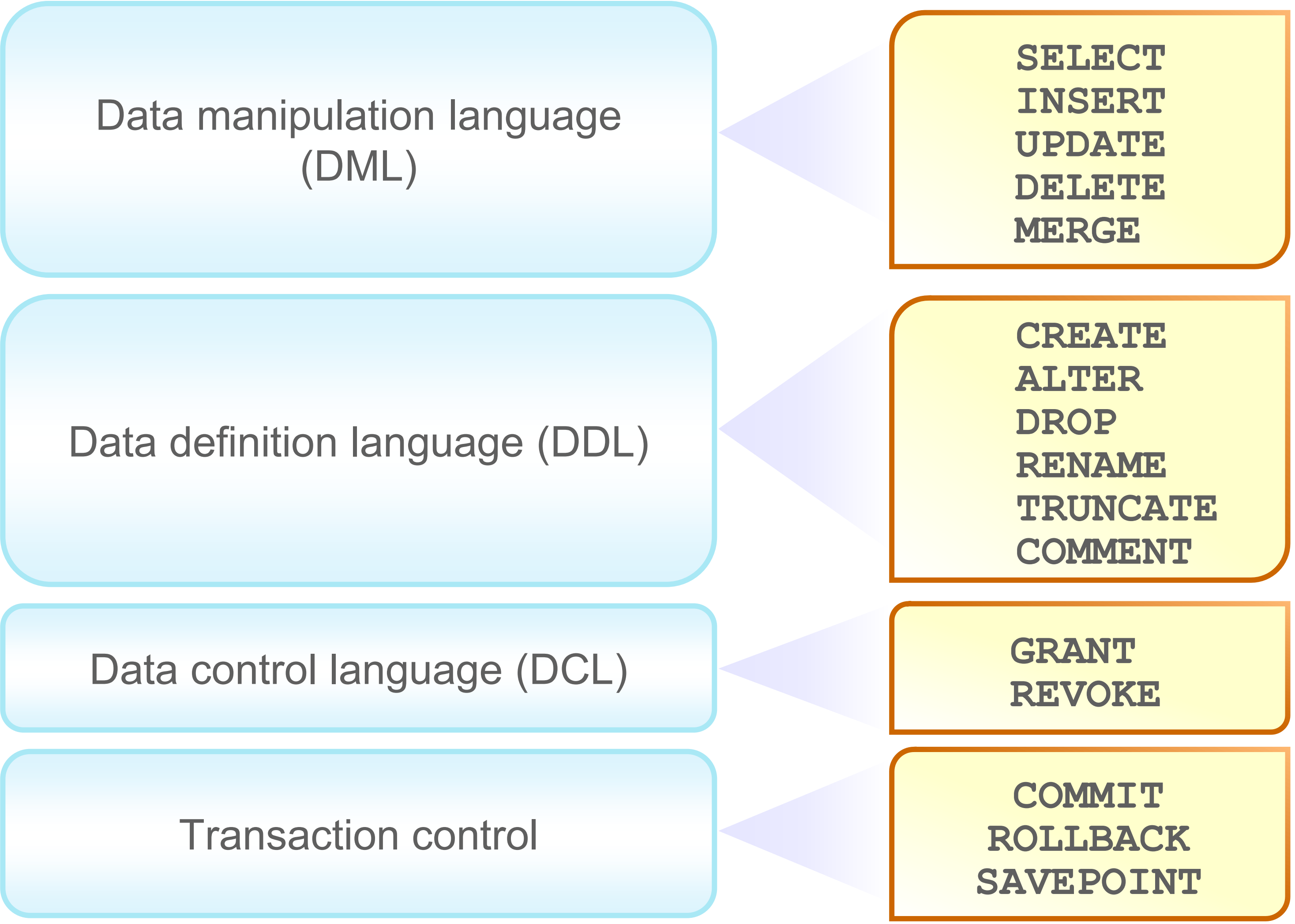


	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

How SQL Works

- SQL is standalone and powerful.
- SQL processes groups of data.
- SQL lets you work with data at a logical level.

SQL Statements Used in the Course



Data manipulation language (DML)	SELECT
	INSERT
	UPDATE
	DELETE
	MERGE

Data definition language (DDL)	CREATE ALTER DROP RENAME TRUNCATE COMMENT
Data control language (DCL)	GRANT REVOKE
Transaction control	COMMIT ROLLBACK SAVEPOINT

Retrieving Data Using the SQL SELECT Statement

Basic SELECT Statement

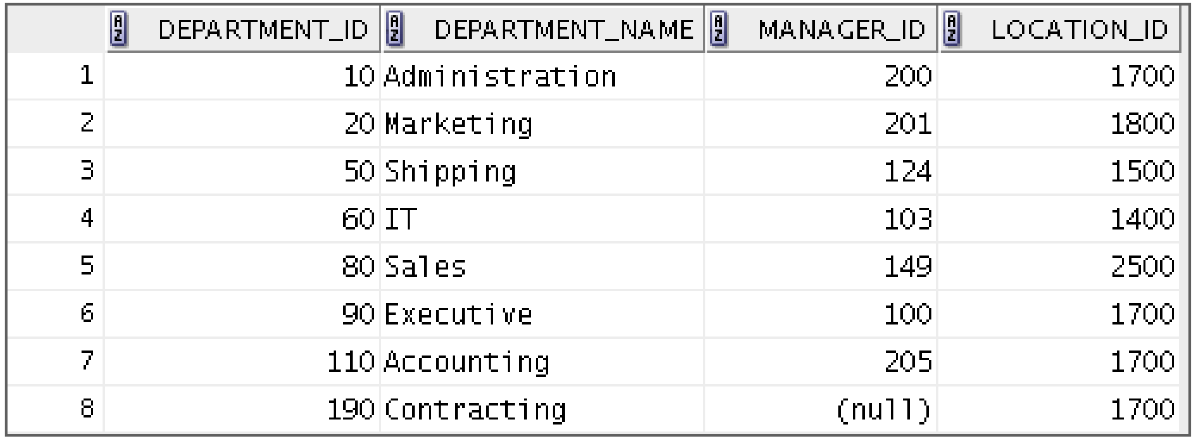
- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

```
SELECT  *|{[DISTINCT] column [alias],...}
FROM    table;
```

Selecting from a table

Selecting All Columns

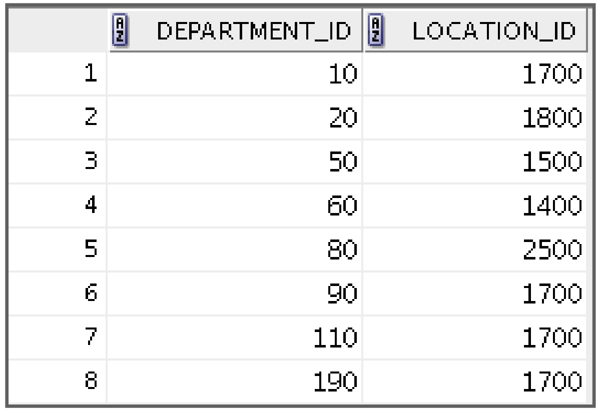
```
SELECT *
FROM   departments;
```



	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Selecting Specific Columns

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



	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

Selecting from DUAL

```
SELECT 5*8;
```

```
SELECT getdate();
```

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

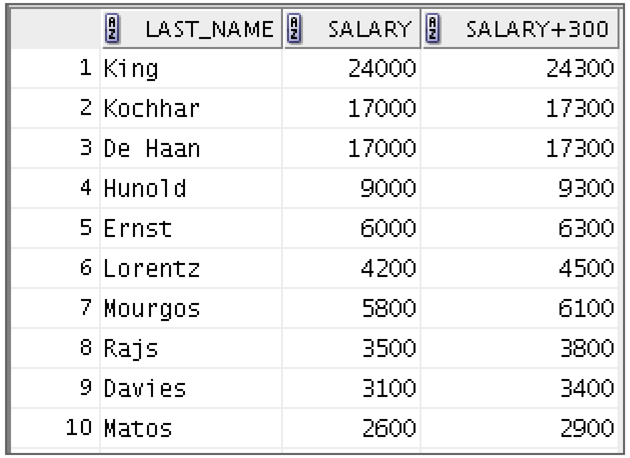
Arithmetic Expressions

You can create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

```
SELECT last_name, salary, salary + 300
FROM   employees;
```



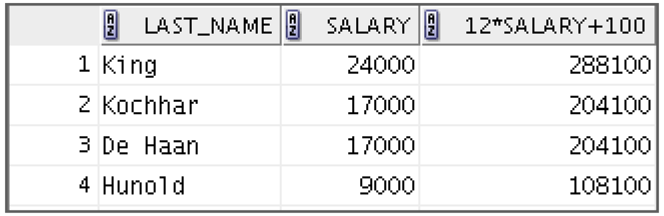
	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...

Operator Precedence

1

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

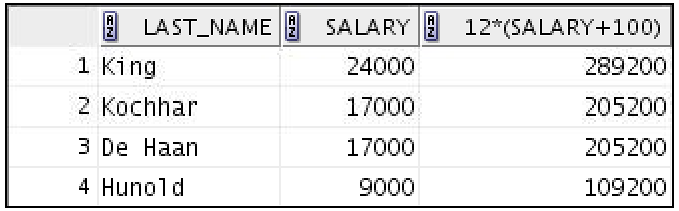


	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

...

2

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```



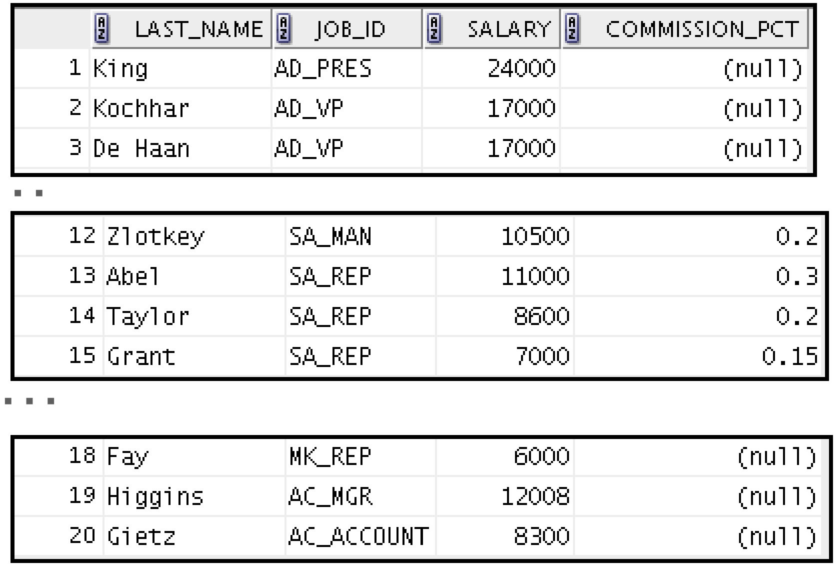
	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200

...

Defining a Null Value

- Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- Null is not the same as zero or a blank space.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

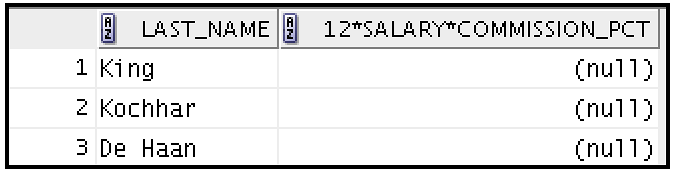


	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	De Haan	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15
...				
18	Fay	MK_REP	6000	(null)
19	Higgins	AC_MGR	12008	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

Null Values in Arithmetic Expressions

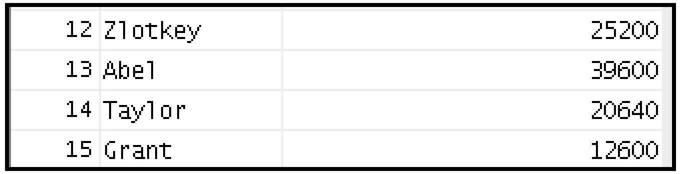
Arithmetic expressions containing a null value evaluate to null.

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



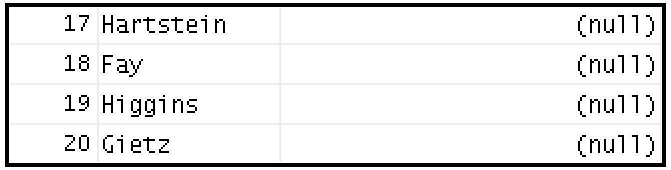
	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...



12	Zlotkey	25200
13	Abel	39600
14	Taylor	20640
15	Grant	12600

...

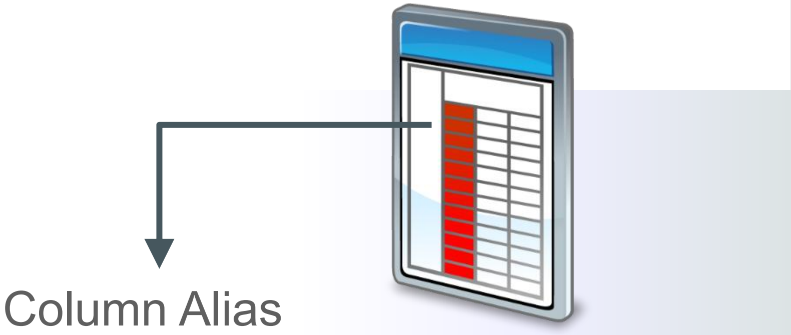


17	Hartstein	(null)
18	Fay	(null)
19	Higgins	(null)
20	Gietz	(null)

Defining a Column Alias

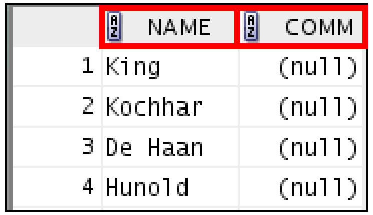
A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name (there can also be the optional AS keyword between the column name and the alias)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive



Using Column Aliases

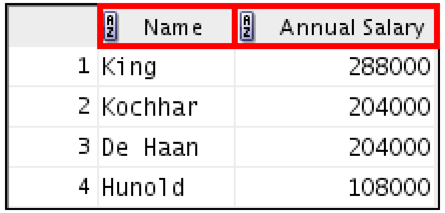
```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```



	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```



	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000

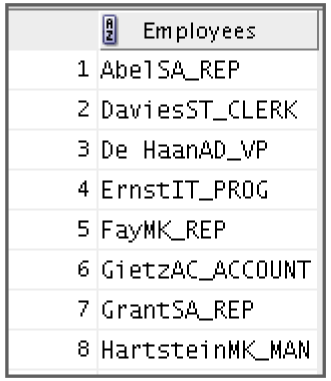
...

Concatenation Operator

The concatenation operator:

- Links columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SELECT last_name + job_id AS "Employees"
FROM   employees;
```



	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP
6	GietzAC_ACCOUNT
7	GrantSA_REP
8	HartsteinMK_MAN

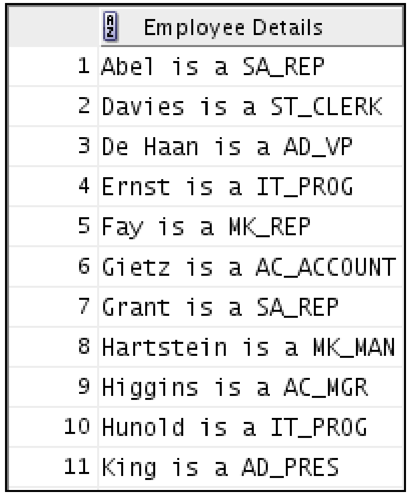
…

Literal Character Strings

- A literal is a character, a number, or a date that is included in the `SELECT` statement.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

Using Literal Character Strings

```
SELECT last_name + ' is a ` + job_id
       AS "Employee Details"
FROM   employees;
```



```
    Employee Details
 1  Abel is a SA_REP
 2  Davies is a ST_CLERK
 3  De Haan is a AD_VP
 4  Ernst is a IT_PROG
 5  Fay is a MK_REP
 6  Gietz is a AC_ACCOUNT
 7  Grant is a SA_REP
 8  Hartstein is a MK_MAN
 9  Higgins is a AC_MGR
10  Hunold is a IT_PROG
11  King is a AD_PRES
```

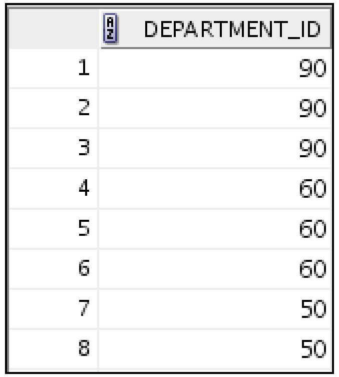
...

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

1

```
SELECT  department_id
FROM    employees;
```

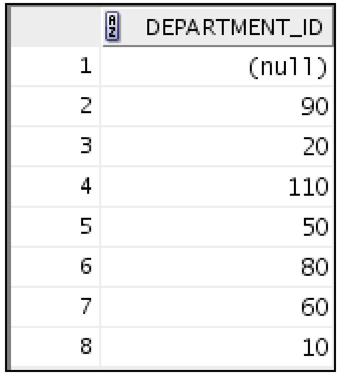


	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

…

2

```
SELECT  DISTINCT department_id
FROM    employees;
```

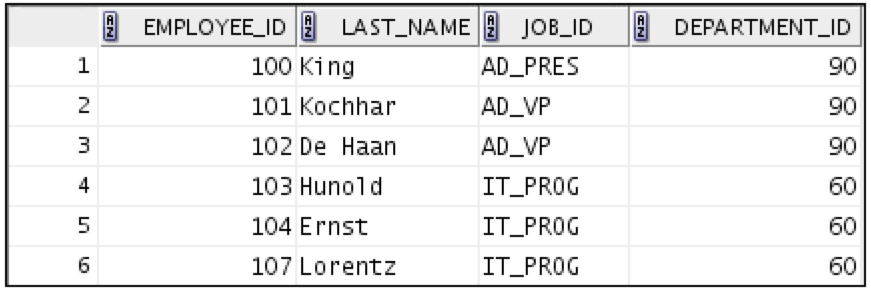


	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10

Restricting and Sorting Data

Limiting Rows by Using a Selection

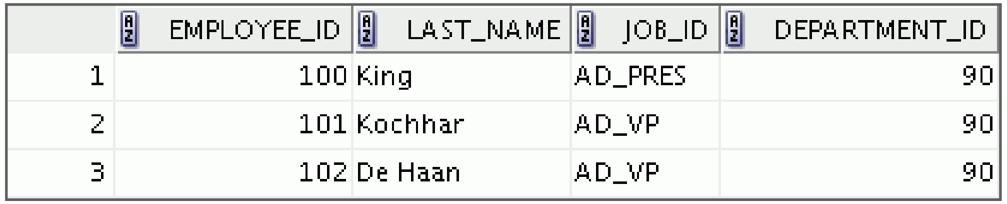
EMPLOYEES



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

…

“retrieve all employees in department 90”



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

Limiting Rows That Are Selected

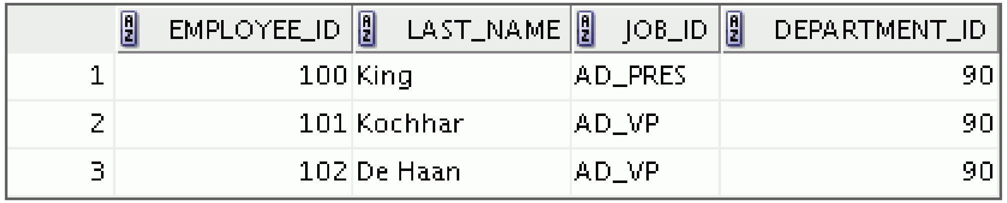
- Restrict the rows that are returned by using the `WHERE` clause:

```
 SELECT   *|{[DISTINCT] column [alias],...}
FROM     table
[WHERE logical expression(s)];
```

- The `WHERE` clause follows the `FROM` clause.

Using the WHERE Clause

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

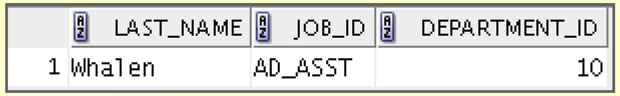


	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

Character Strings and Dates

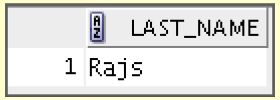
- Character strings and date values are enclosed within single quotation marks (`''`).
- Character values are case-sensitive and date values are format-sensitive.
- The default display format for date is `DD-MON-RR`.

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



	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-OCT-11' ;
```



	LAST_NAME
1	Rajs

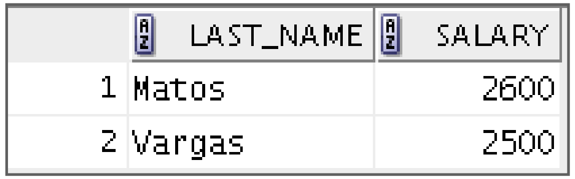
Comparison Operators

Operator	Meaning
`=`	Equal to
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to
`<>`	Not equal to
`BETWEEN ...AND...`	Between two values (inclusive)
`IN(set)`	Match any of a list of values
`LIKE`	Match a character pattern
`IS NULL`	Is a null value

Using Comparison Operators

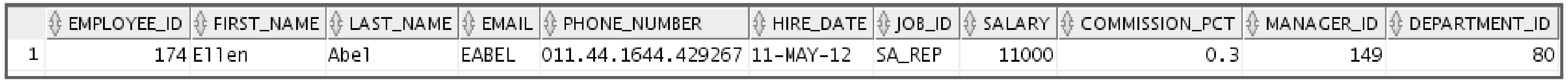
Let us look at some examples:

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



	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

```
SELECT *
FROM   employees
WHERE  last_name = 'Abel';
```



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-12	SA_REP	11000	0.3	149	80

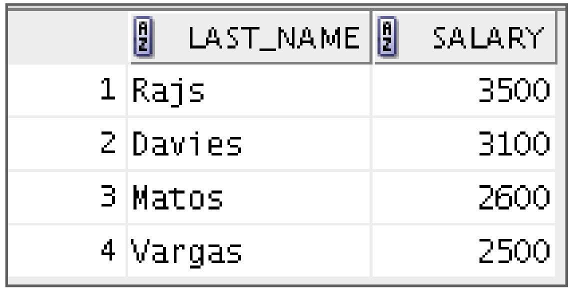
Range Conditions Using the BETWEEN Operator

You can use the BETWEEN operator to display rows based on a range of values:

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

Lower limit

Upper limit

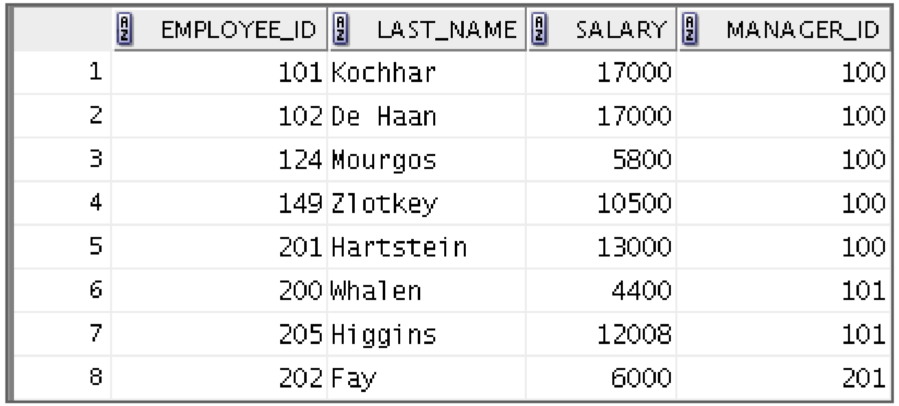


	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

Using the IN Operator

Use the IN operator to test for values in a list:

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

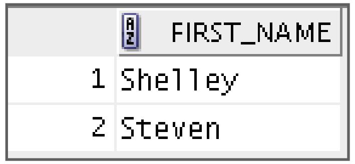


	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

Pattern Matching Using the LIKE Operator

- You can use the LIKE operator to perform wildcard searches of valid string patterns.
- The search conditions can contain either literal characters or numbers:
  - % denotes zero or more characters.
  - _ denotes one character.

```
SELECT first_name
FROM   employees
WHERE  first_name LIKE 'S%' ;
```

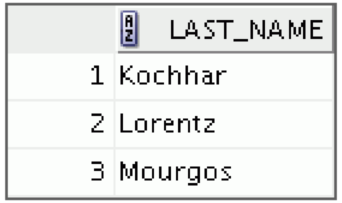


	FIRST_NAME
1	Shelley
2	Steven

Combining Wildcard Symbols

- You can combine the two wildcard symbols (%, _) with literal characters for pattern matching:

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



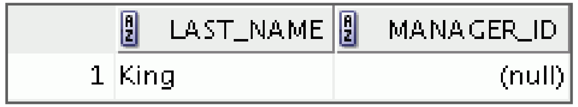
	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

- You can use the ESCAPE identifier to search for the actual % and _ symbols.

Using NULL Conditions

You can use the IS NULL operator to test for NULL values in a column.

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



	LAST_NAME	MANAGER_ID
1	King	(null)

Defining Conditions Using Logical Operators

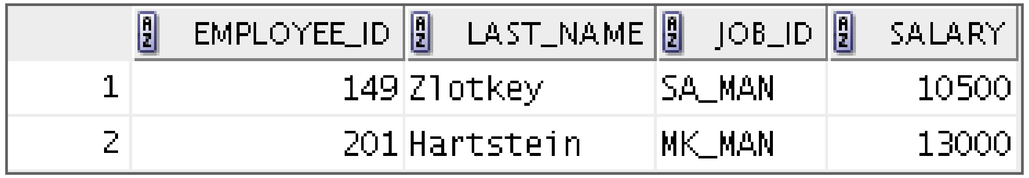
You can use the logical operators to filter the result set based on more than one condition or invert the result set.

Operator	Meaning
`AND`	Returns `TRUE` if *both* component conditions are true
`OR`	Returns `TRUE` if *either* component condition is true
`NOT`	Returns `TRUE` if the condition is false

Using the AND Operator

AND requires both the component conditions to be true:

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

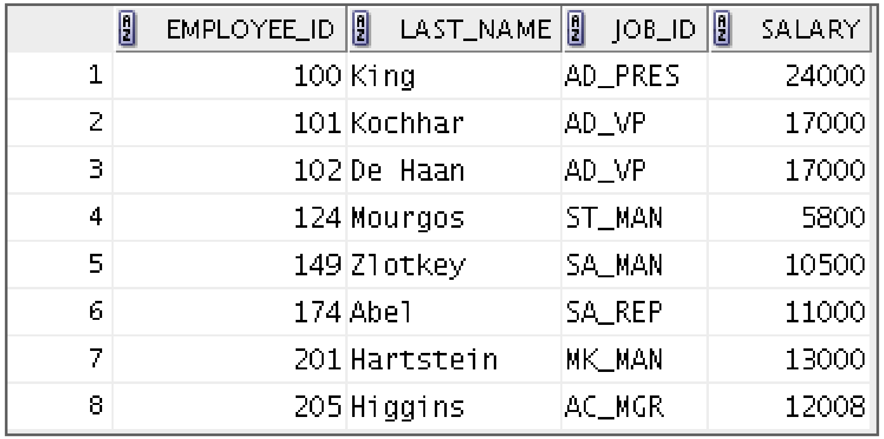


	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

Using the OR Operator

OR requires either component condition to be true:

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

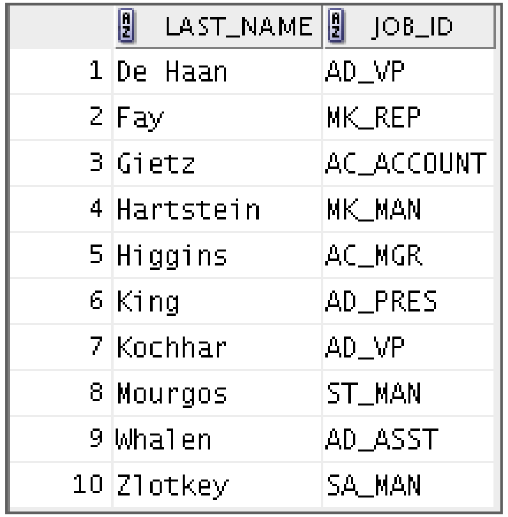


	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12008

Using the NOT Operator

NOT is used to negate a condition:

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



	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

Rules of Precedence

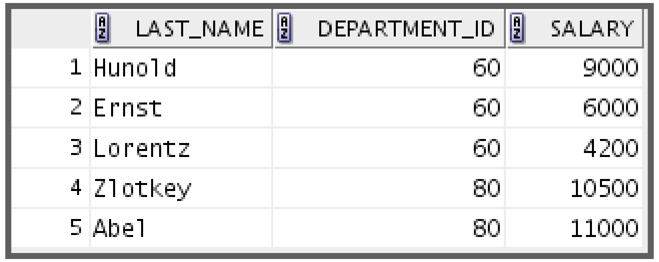
Order	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical operator
8	AND logical operator
9	OR logical operator

You can use parentheses to override rules of precedence.

Rules of Precedence

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  department_id = 60
OR     department_id = 80
AND    salary > 10000;
```

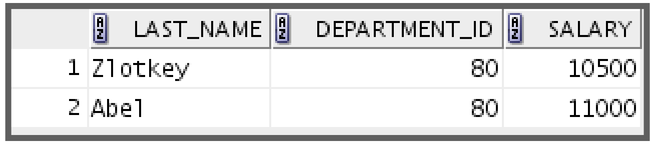
1



	LAST_NAME	DEPARTMENT_ID	SALARY
1	Hunold	60	9000
2	Ernst	60	6000
3	Lorentz	60	4200
4	Zlotkey	80	10500
5	Abel	80	11000

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (department_id = 60
OR     department_id = 80)
AND    salary > 10000;
```

2



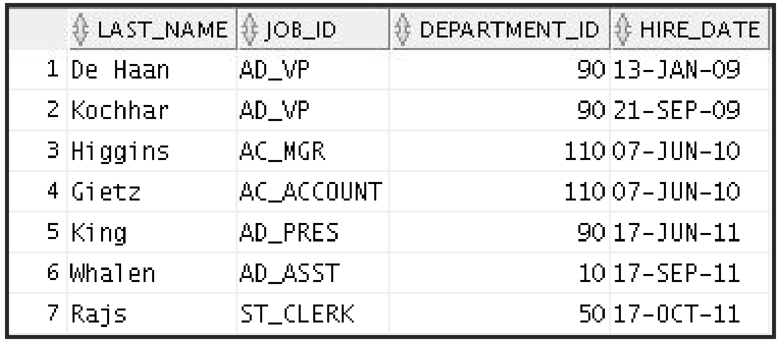
	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

Using the ORDER BY Clause

You can sort the retrieved rows with the ORDER BY clause:

- ASC: Ascending order, default
- DESC: Descending order

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



	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP	90	13-JAN-09
2	Kochhar	AD_VP	90	21-SEP-09
3	Higgins	AC_MGR	110	07-JUN-10
4	Gietz	AC_ACCOUNT	110	07-JUN-10
5	King	AD_PRES	90	17-JUN-11
6	Whalen	AD_ASST	10	17-SEP-11
7	Rajs	ST_CLERK	50	17-OCT-11

…

Sorting

- Sorting in descending order:

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

1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal ;
```

2

Sorting

- Sorting by using the column’s numeric position:

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY 3;
```

3

- Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

4

SQL Row Limiting Clause

- You can use the `row_limiting_clause` to limit the rows that are returned by a query.
- You can use this clause to implement Top-N reporting.

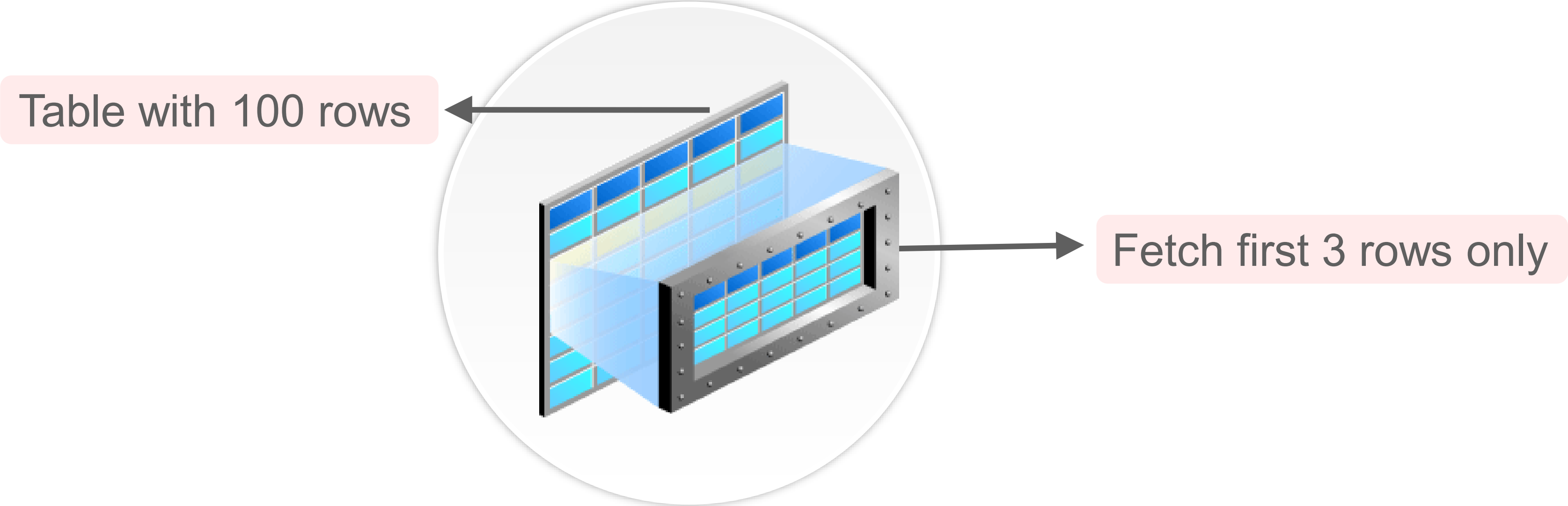
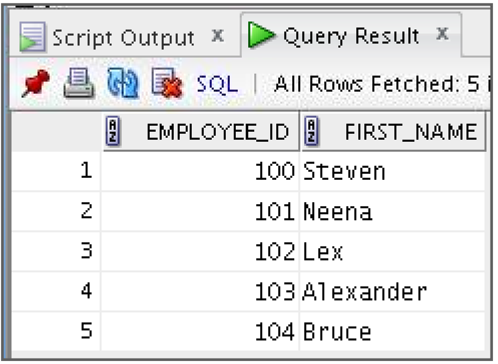


Table with 100 rows

Fetch first 3 rows only

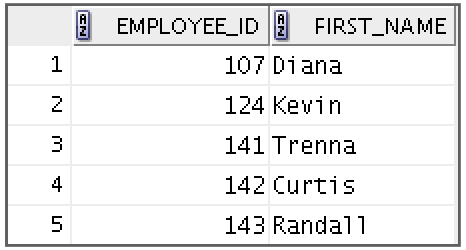
SQL Row Limiting Clause: Example

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 0 ROWS FETCH FIRST 5 ROWS ONLY;
```



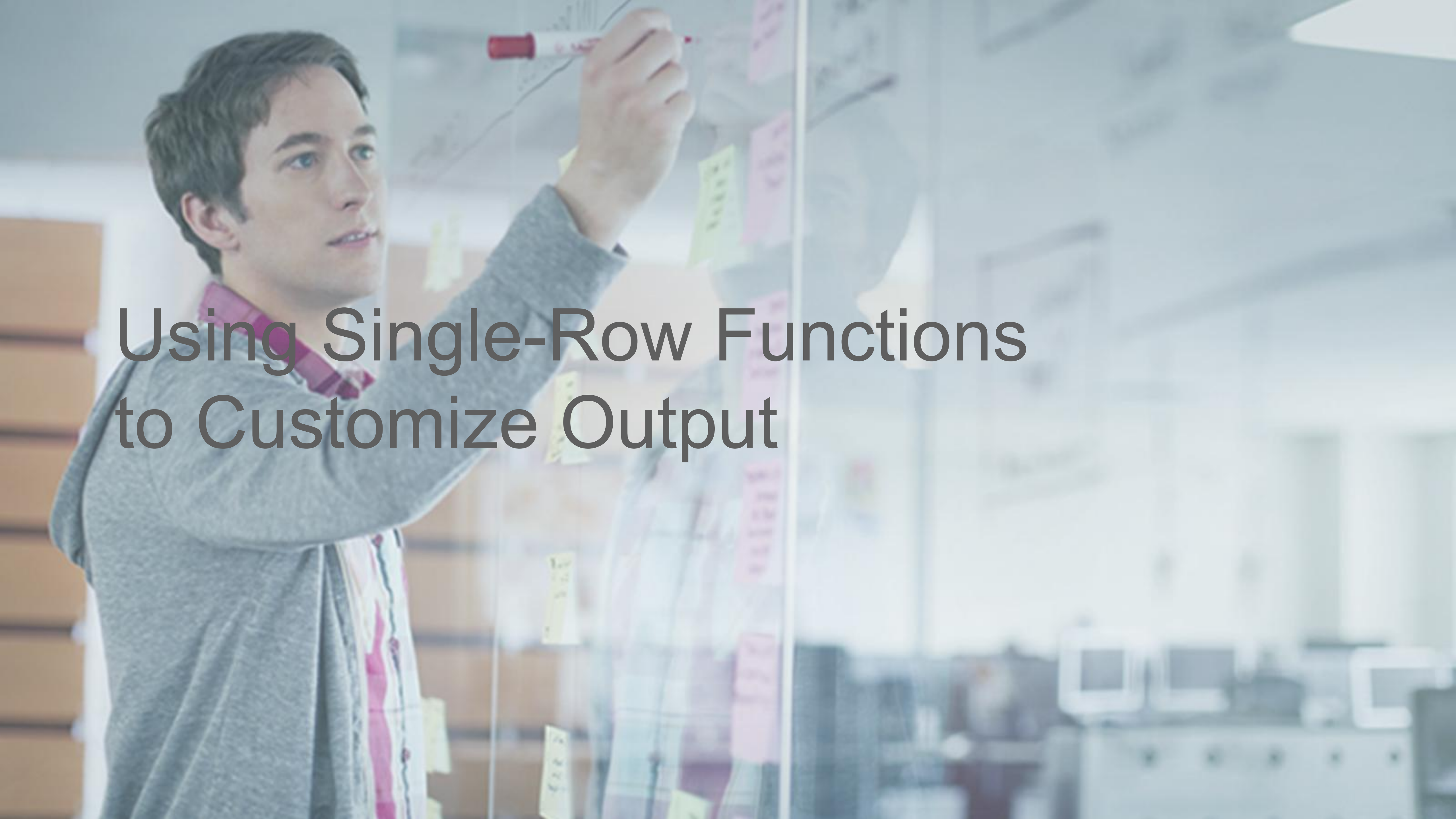
	EMPLOYEE_ID	FIRST_NAME
1	100	Steven
2	101	Neena
3	102	Lex
4	103	Alexander
5	104	Bruce

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

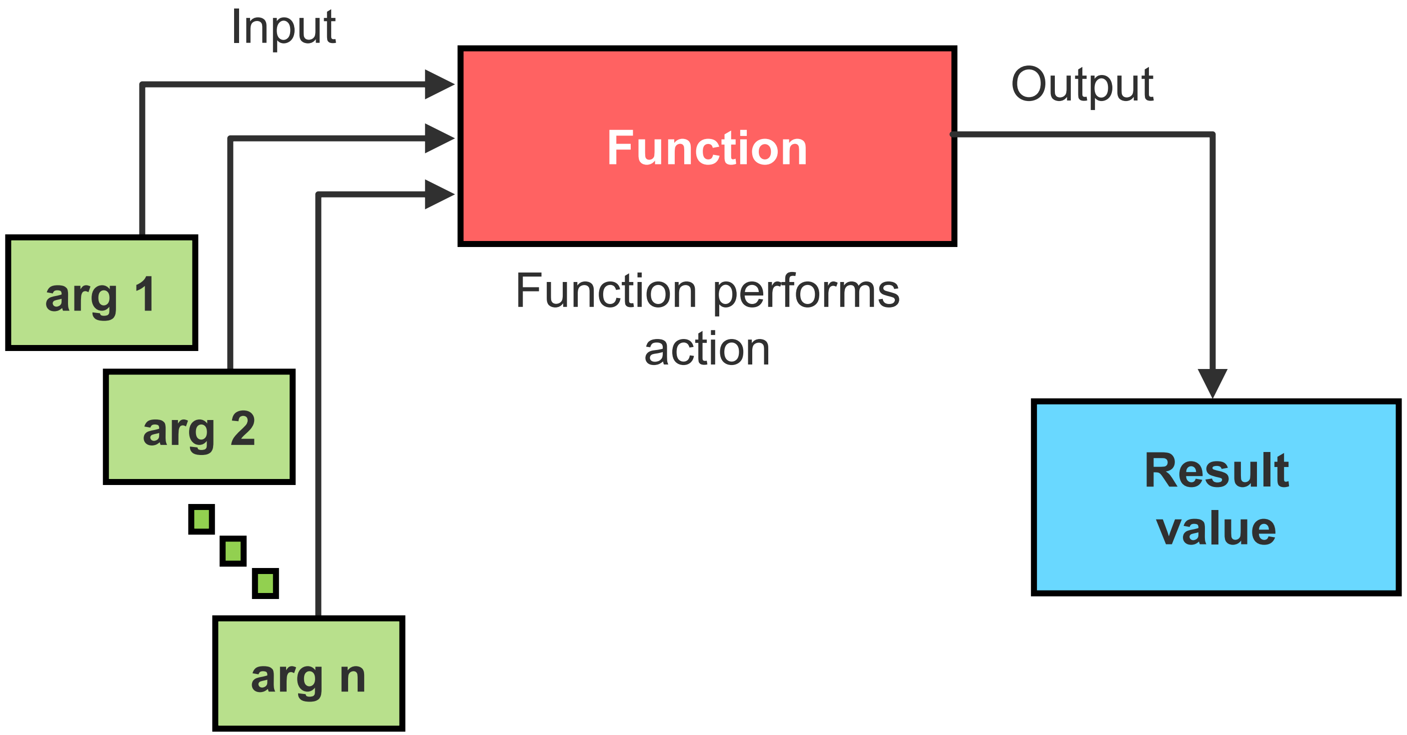


	EMPLOYEE_ID	FIRST_NAME
1	107	Diana
2	124	Kevin
3	141	Trenna
4	142	Curtis
5	143	Randall

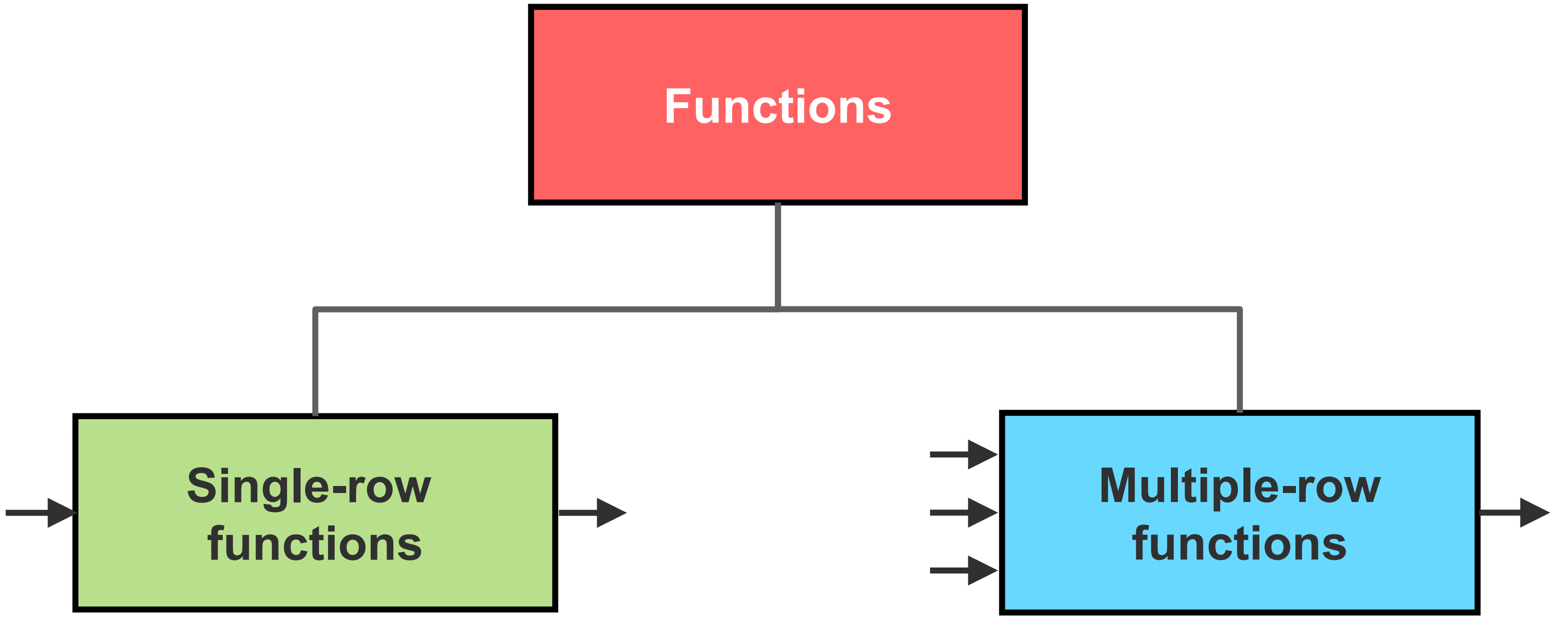
Using Single-Row Functions to Customize Output



SQL Functions



Two Types of SQL Functions



Functions

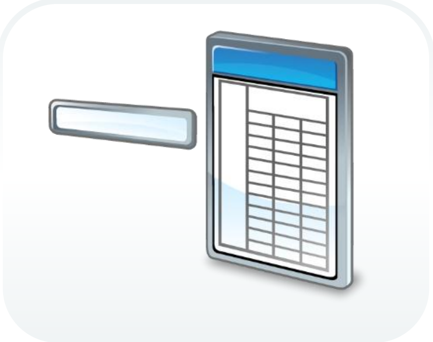
Single-row functions

Multiple-row functions

Returns one result per row

Returns one result per set of rows

Single-Row Functions

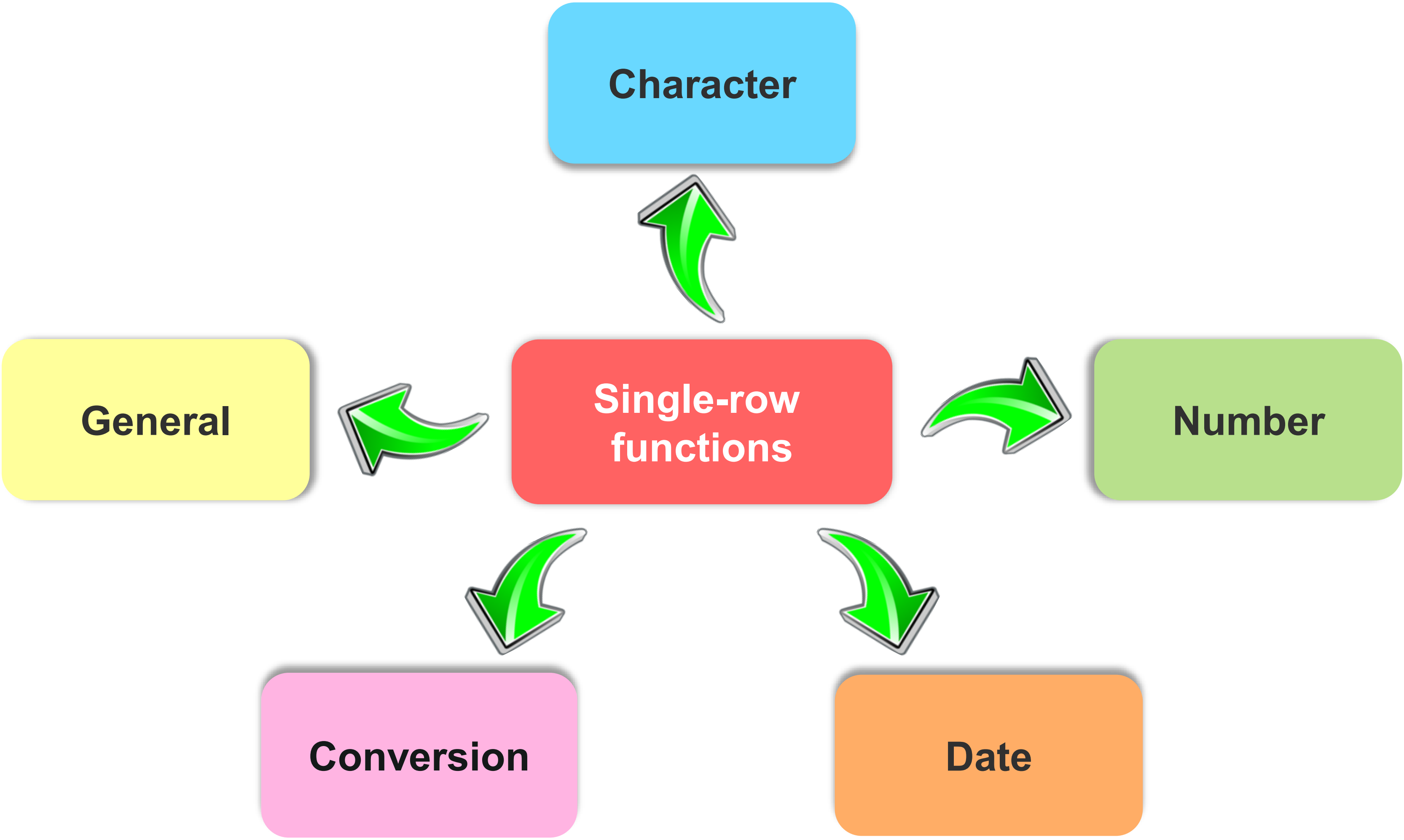


Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2,...)]
```

Single-Row Functions



Character

General

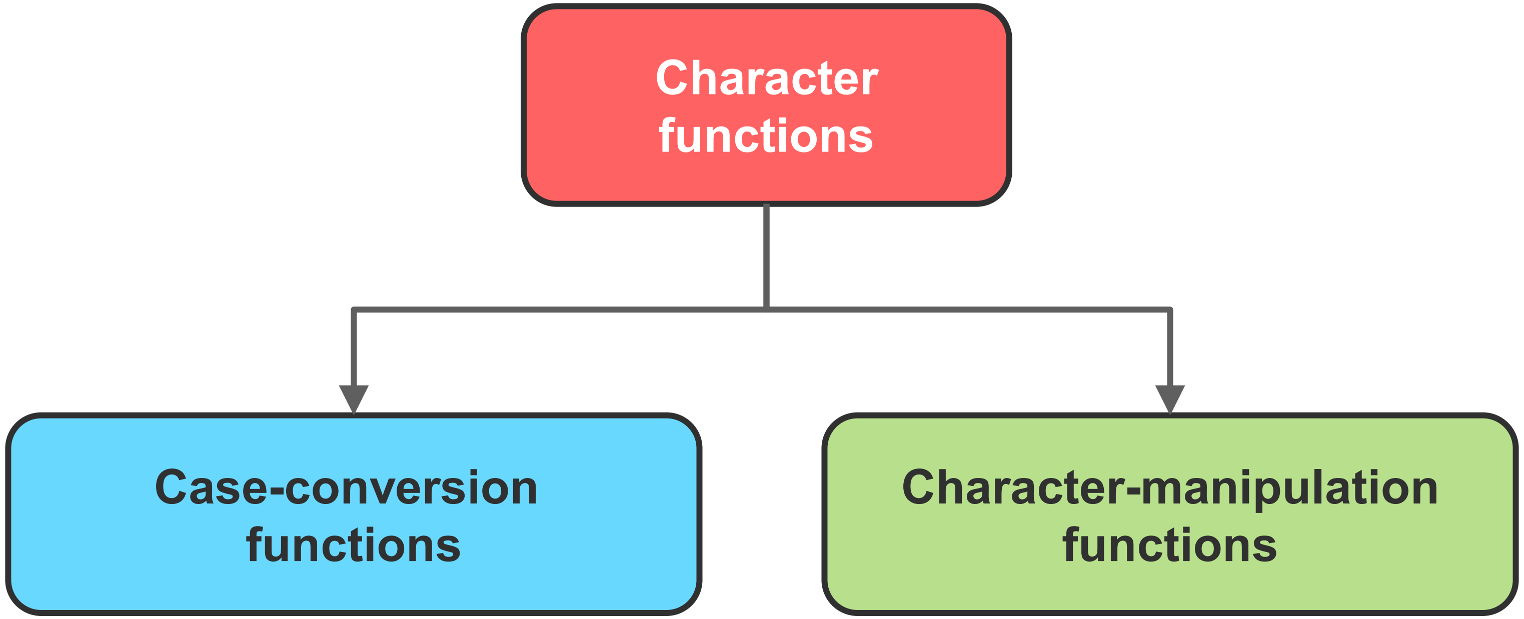
Single-row functions

Number

Conversion

Date

Character Functions



Character functions

Case-conversion functions

Character-manipulation functions

```
LOWER
UPPER
```

```
CONCAT
SUBSTRING
LEN
INSTR
TRIM
REPLACE
```

Case-Conversion Functions

You can use these functions to convert the case of character strings:

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

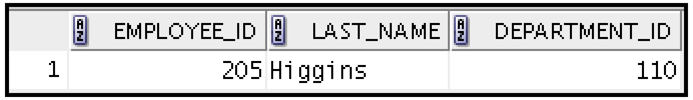
Using Case-Conversion Functions

Display the employee number, name, and department number for employee Higgins:

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

```
0 rows selected
```

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



	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

Character-Manipulation Functions

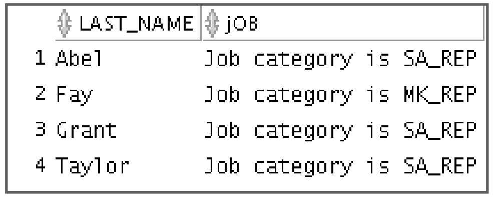
You can use these functions to manipulate character strings:

Function	Result
`CONCAT('Hello', 'World')`	`HelloWorld`
`SUBSTRING('HelloWorld',1,5)`	`Hello`
`LEN('HelloWorld')`	`10`
`CHARINDEX('HelloWorld', 'W')`	`6`

Using Character-Manipulation Functions

```
SELECT last_name, CONCAT('Job category is ', job_id)
"Job" FROM   employees
WHERE  SUBSTRING (job_id, 4,3) = 'REP';
```

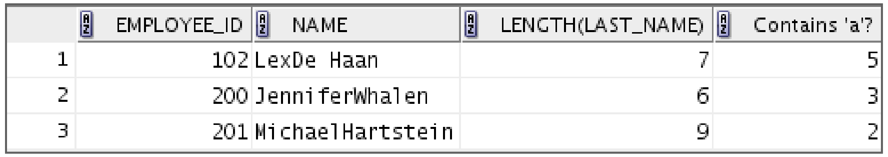
1



	LAST_NAME	JOB
1	Abel	Job category is SA_REP
2	Fay	Job category is MK_REP
3	Grant	Job category is SA_REP
4	Taylor	Job category is SA_REP

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LEN (last_name), CHARINDEX('a',last_name) "Contains 'a'?"
FROM   employees
WHERE  SUBSTRING (last_name, len(last_name),1) = 'n';
```

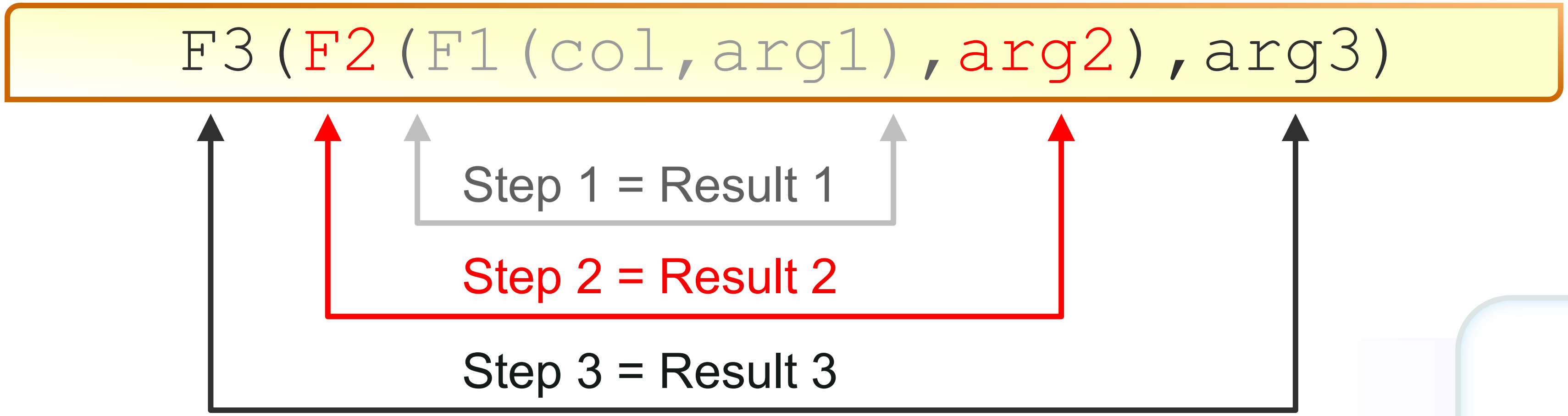
2



	EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	102	LexDe Haan	7	5
2	200	JenniferWhalen	6	3
3	201	MichaelHartstein	9	2

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from the deepest level to the least deep level.

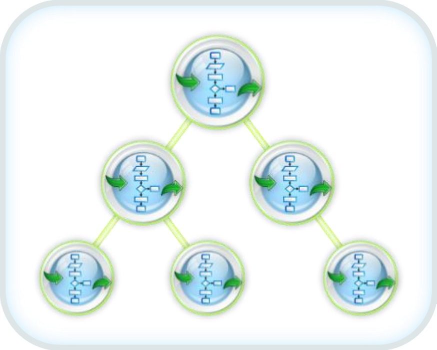


```
F3(F2(F1(col,arg1),arg2),arg3)
```

Step 1 = Result 1

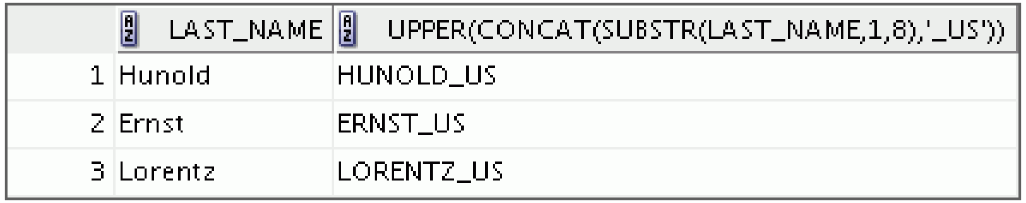
Step 2 = Result 2

Step 3 = Result 3



Nesting Functions: Example

```
SELECT last_name,
  UPPER(CONCAT(SUBSTRING  (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department_id = 60;
```



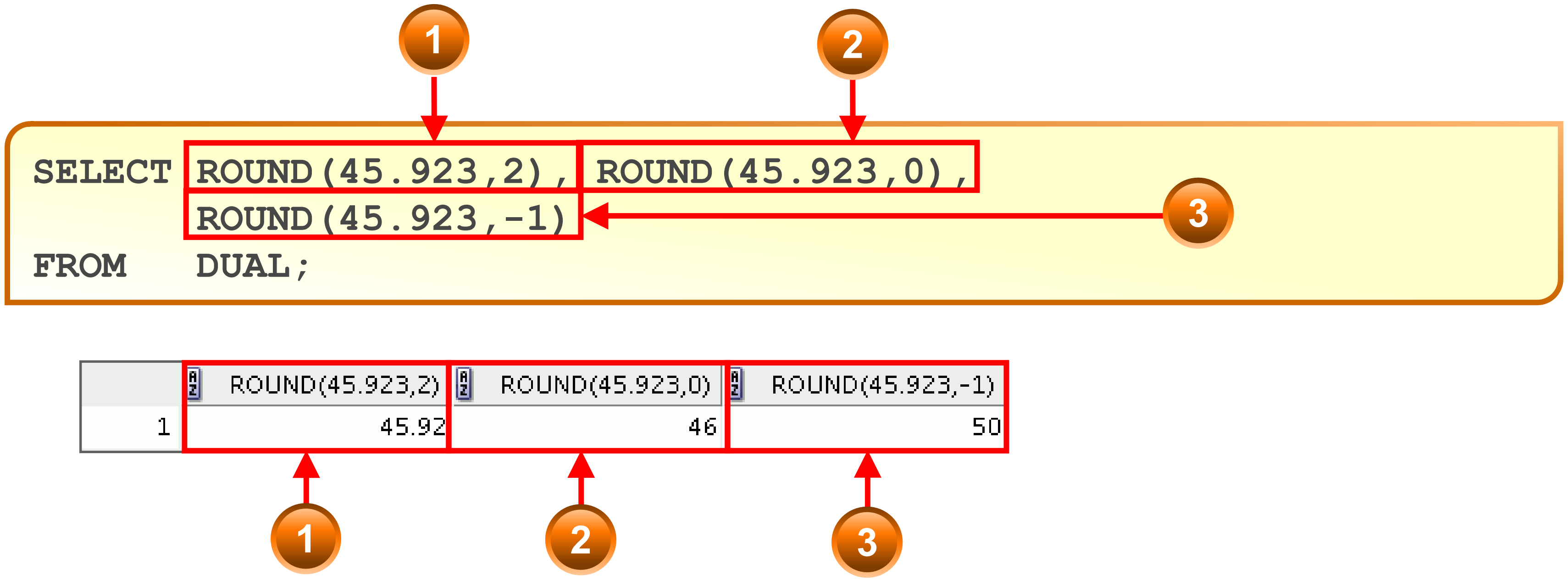
	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

Numeric Functions

- `ROUND`: Rounds value to a specified decimal
- `CEIL`: Returns the smallest whole number greater than or equal to a specified number
- `FLOOR`: Returns the largest whole number equal to or less than a specified number

Function	Result
`ROUND(45.926, 2)`	`45.93`
`CEIL (2.83)`	`3`
`FLOOR (2.83)`	`2`

Using the ROUND Function



```
SELECT ROUND(45.923,2), ROUND(45.923,0),
       ROUND(45.923,-1)
FROM   DUAL;
```

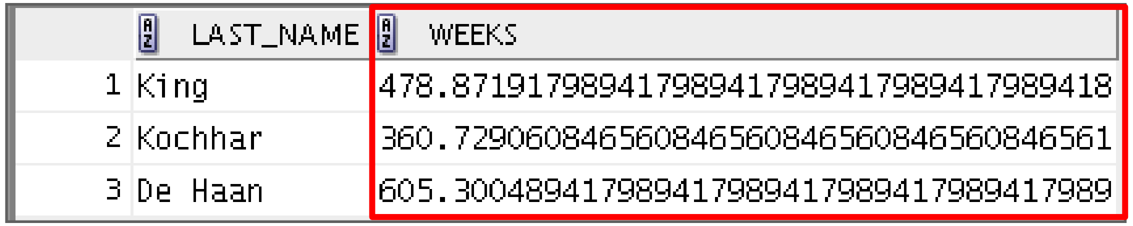
	ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
1	45.92	46	50

Arithmetic with Dates

- Add to or subtract a number from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

Using Arithmetic Operators with Dates

```
SELECT  datediff(day, hire_date,GETDATE())
from employees
```



	LAST_NAME	WEEKS
1	King	478.871917989417989417989417989417989418
2	Kochhar	360.729060846560846560846560846560846561
3	De Haan	605.300489417989417989417989417989417989

Date-Manipulation Functions

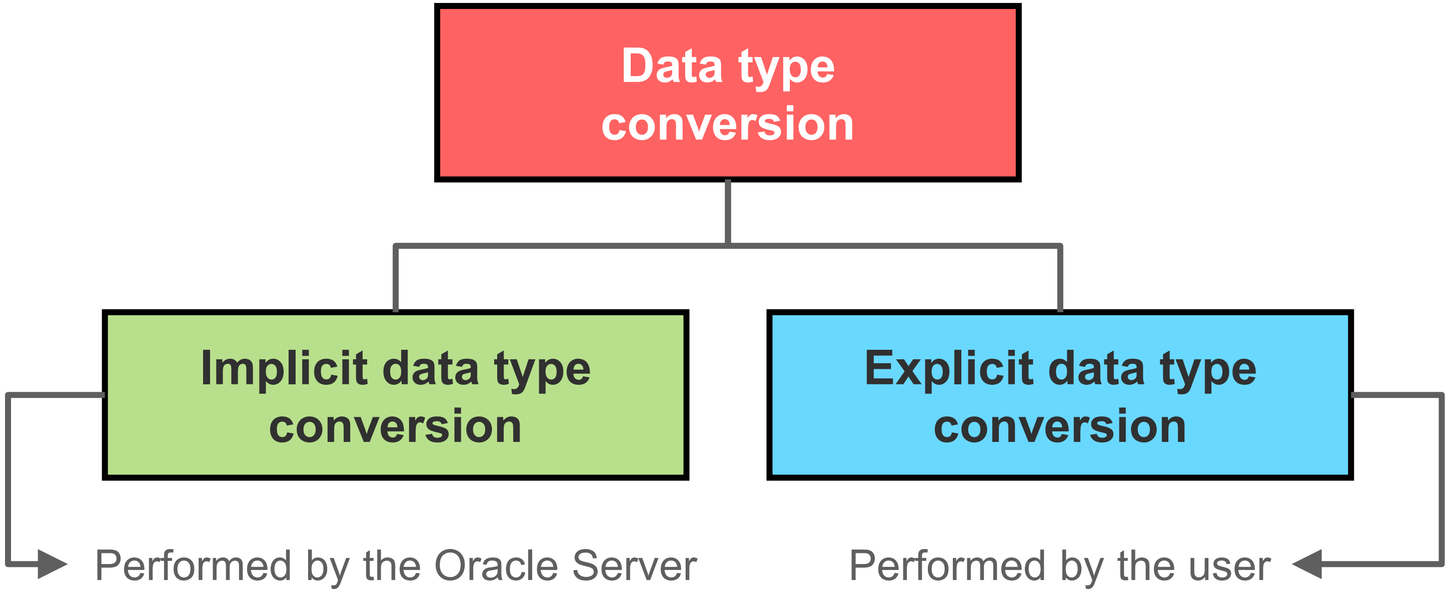
Function	Result
DATEDIFF	Number of months between two dates
DATEADD	Add calendar months to date
ROUND	Round date

Using Date Functions

Function	Result
`DATEDIFF(month, '2013-02-28', '2013-03-01');`	`1`
`ADD_MONTHS ('31-JAN-16',1)`	`'29-FEB-16'`
`EOMONTH(‘01-APR-16')`	`'30-APR-16'`

Using Conversion Functions and Conditional Expressions

Conversion Functions



Data type conversion

Implicit data type conversion

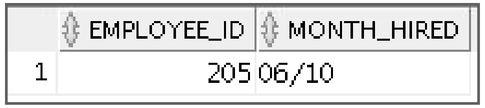
Explicit data type conversion

Performed by the Oracle Server

Performed by the user

Example:

```
SELECT employee_id, format(hire_date, 'dd-MM-yyyy')
Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```



	EMPLOYEE_ID	MONTH_HIRED
1	205	06/10

ISNULL Function

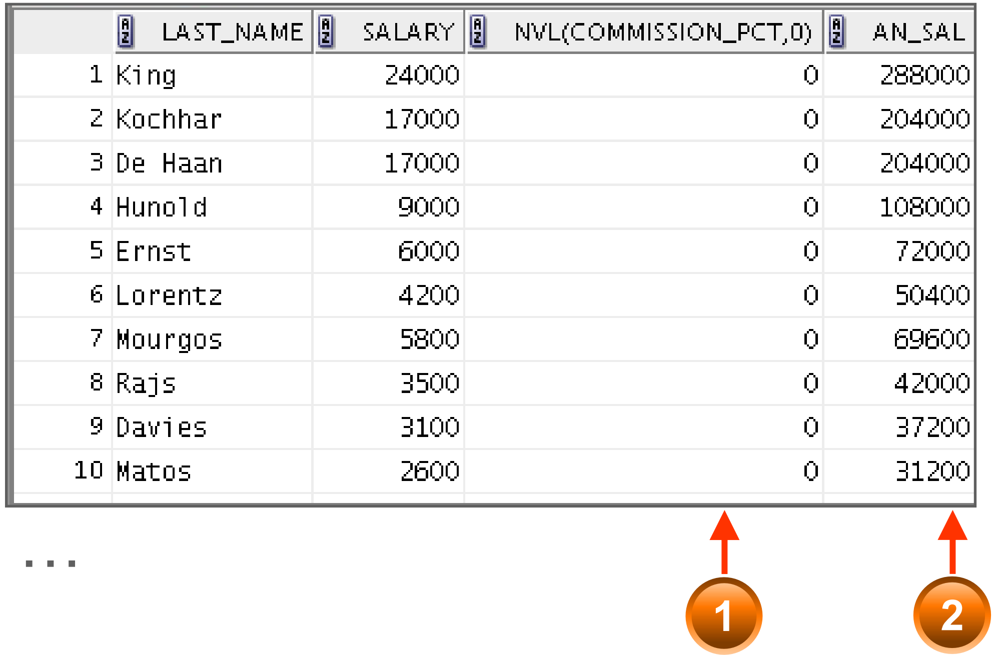
Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match.
- Examples:
  - `ISNULL(commission_pct,0)`
  - `ISNULL(hire_date,'01-JAN-97')`
  - `ISNULL(job_id,'No Job Yet')`

NVL (expr1, expr2)

Using the ISNULL Function

```
SELECT last_name, salary, ISNULL(commission_pct, 0),
   (salary*12) + (salary*12*ISNULL (commission_pct, 0)) AN_SAL
FROM employees;
```



	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200

...

Using the NULLIF Function

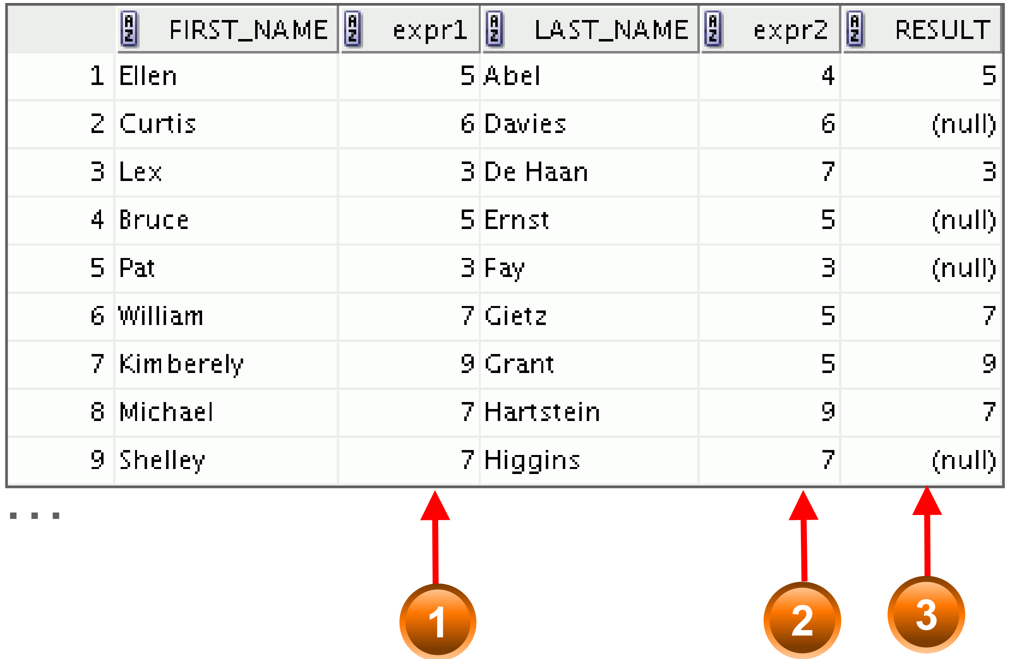
```
NULLIF (expr1,
expr2)
```

```
SELECT first_name, LEN(first_name) "expr1",
       last_name,  LEN(last_name)  "expr2",
       NULLIF(LEN(first_name), LEN(last_name)) result
FROM   employees;
```

1

2

3



	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis	6	Davies	6	(null)
3	Lex	3	De Haan	7	3
4	Bruce	5	Ernst	5	(null)
5	Pat	3	Fay	3	(null)
6	William	7	Gietz	5	7
7	Kimberely	9	Grant	5	9
8	Michael	7	Hartstein	9	7
9	Shelley	7	Higgins	7	(null)

...

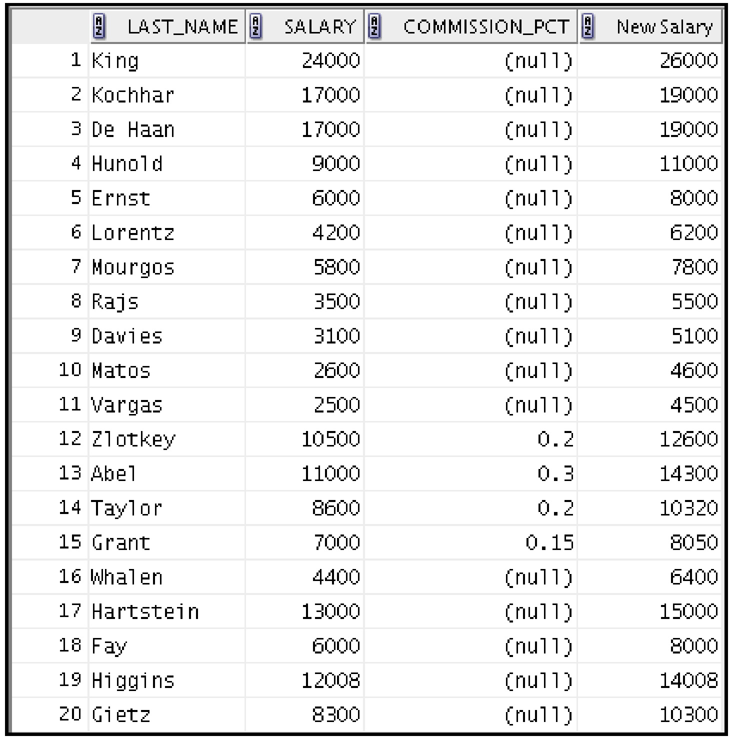
Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternative values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.

```
COALESCE (expr1, expr2, ..., exprn)
```

Using the COALESCE Function

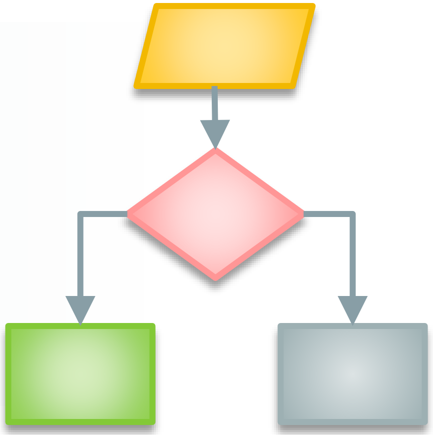
```
SELECT last name, salary, commission pct,
COALESCE((salary+(commission pct*salary)), salary+2000)"New
Salary"
FROM   employees;
```



	LAST_NAME	SALARY	COMMISSION_PCT	New Salary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000
5	Ernst	6000	(null)	8000
6	Lorentz	4200	(null)	6200
7	Mourgos	5800	(null)	7800
8	Rajs	3500	(null)	5500
9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12008	(null)	14008
20	Gietz	8300	(null)	10300

Conditional Expressions

- Help provide the use of `IF-THEN-ELSE` logic within a SQL statement
- You can use the following methods:
  - `CASE` expression
  - Searched `CASE` expression
  - `DECODE` function



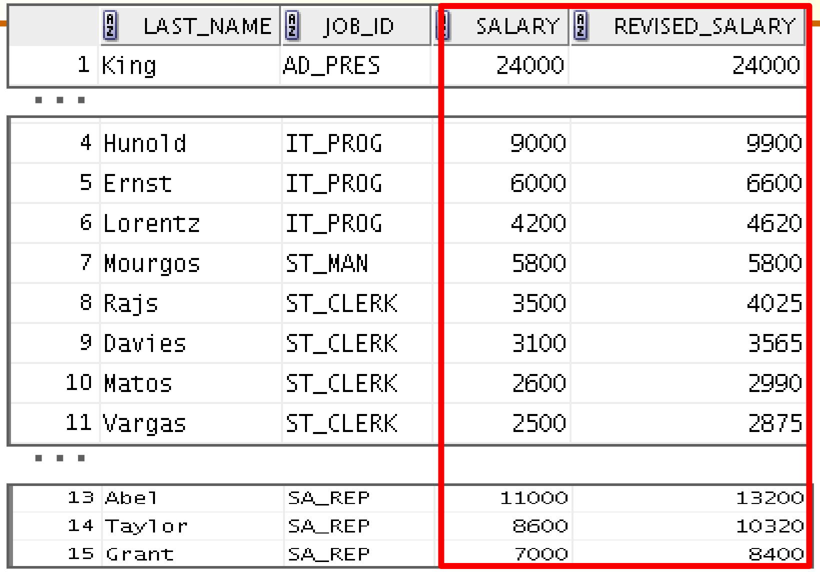
CASE Expression

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

```
CASE expr WHEN comparison_expr1 THEN return_expr1
         [WHEN comparison_expr2 THEN return_expr2
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END
```

Using the CASE Expression

```
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;
```



	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	King	AD_PRES	24000	24000
...				
4	Hunold	IT_PROG	9000	9900
5	Ernst	IT_PROG	6000	6600
6	Lorentz	IT_PROG	4200	4620
7	Mourgos	ST_MAN	5800	5800
8	Rajs	ST_CLERK	3500	4025
9	Davies	ST_CLERK	3100	3565
10	Matos	ST_CLERK	2600	2990
11	Vargas	ST_CLERK	2500	2875
...				
13	Abel	SA_REP	11000	13200
14	Taylor	SA_REP	8600	10320
15	Grant	SA_REP	7000	8400

Searched CASE Expression

```
CASE
     WHEN condition1 THEN use_expression1
     WHEN condition2 THEN use_expression2
     WHEN condition3 THEN use_expression3
     ELSE default_use_expression
END
```

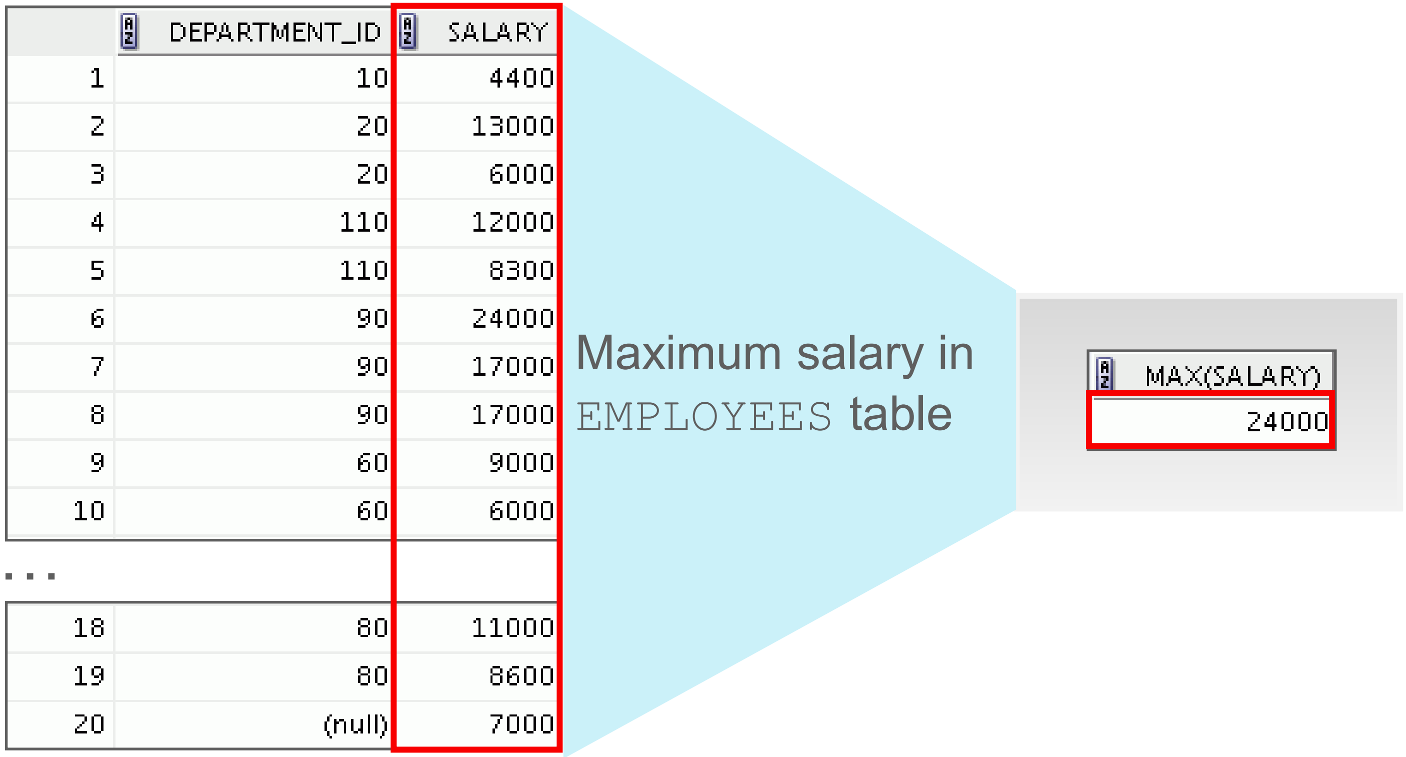
```
SELECT last name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

Reporting Aggregated Data Using the Group Functions

Group Functions

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

EMPLOYEES



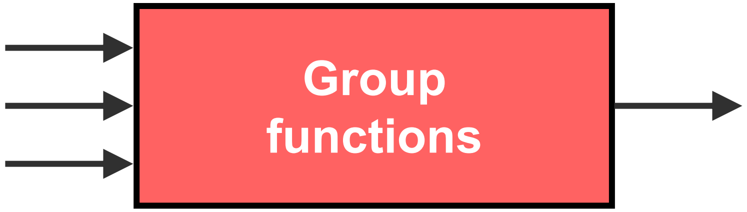
	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
…		
18	80	11000
19	80	8600
20	(null)	7000

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

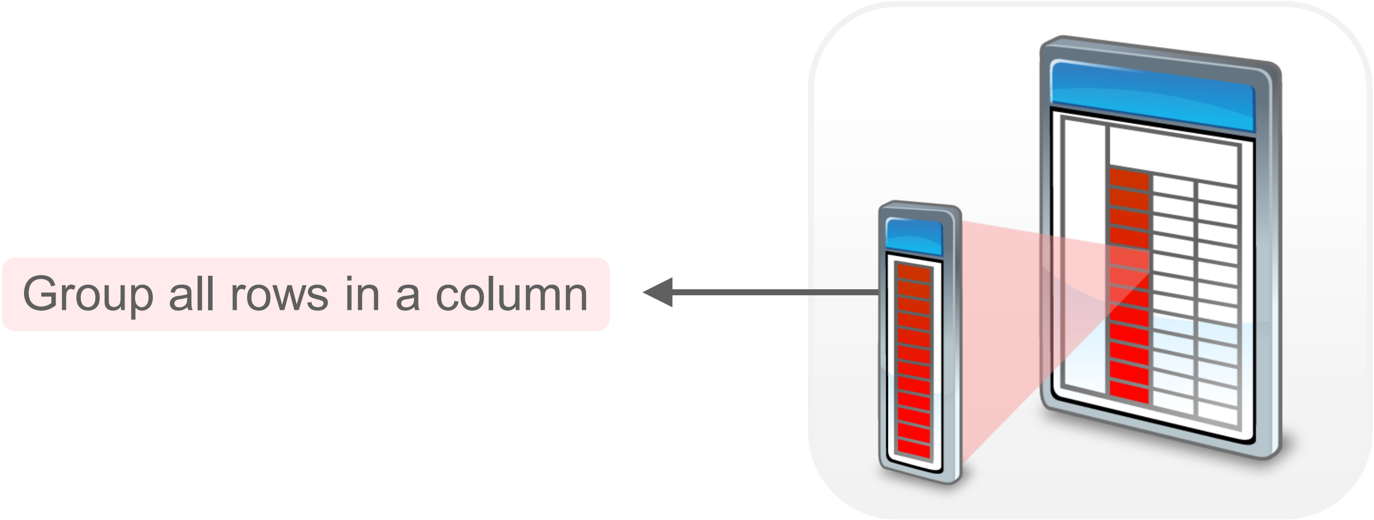
Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



Group Functions: Syntax

```
SELECT    group_function(column), ...
FROM      table
[WHERE    condition];
```

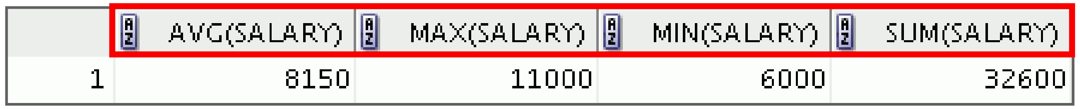


Group all rows in a column

Using the AVG and SUM Functions

You can use the AVG and SUM functions for numeric data.

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

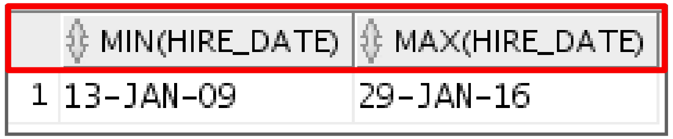


	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

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

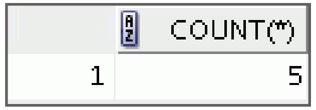


```
    MIN(HIRE_DATE)  MAX(HIRE_DATE)
1 13-JAN-09       29-JAN-16
```

Using the COUNT Function

1

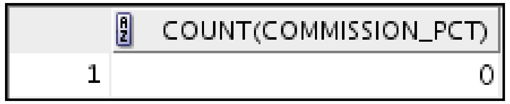
```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```



	COUNT(*)
1	5

2

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

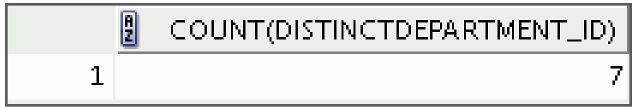


	COUNT(COMMISSION_PCT)
1	0

Using the DISTINCT Keyword

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

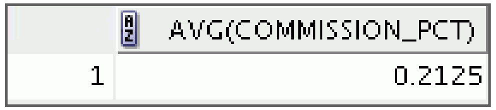


```
     COUNT(DISTINCTDEPARTMENT_ID)
1                               7
```

Group Functions and Null Values

```
SELECT AVG(commission_pct)
FROM   employees;
```

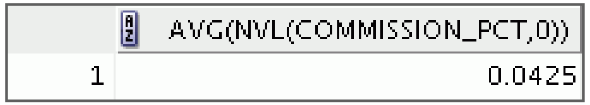
1



	AVG(COMMISSION_PCT)
1	0.2125

```
SELECT AVG(isnull(commission_pct, 0))
FROM   employees;
```

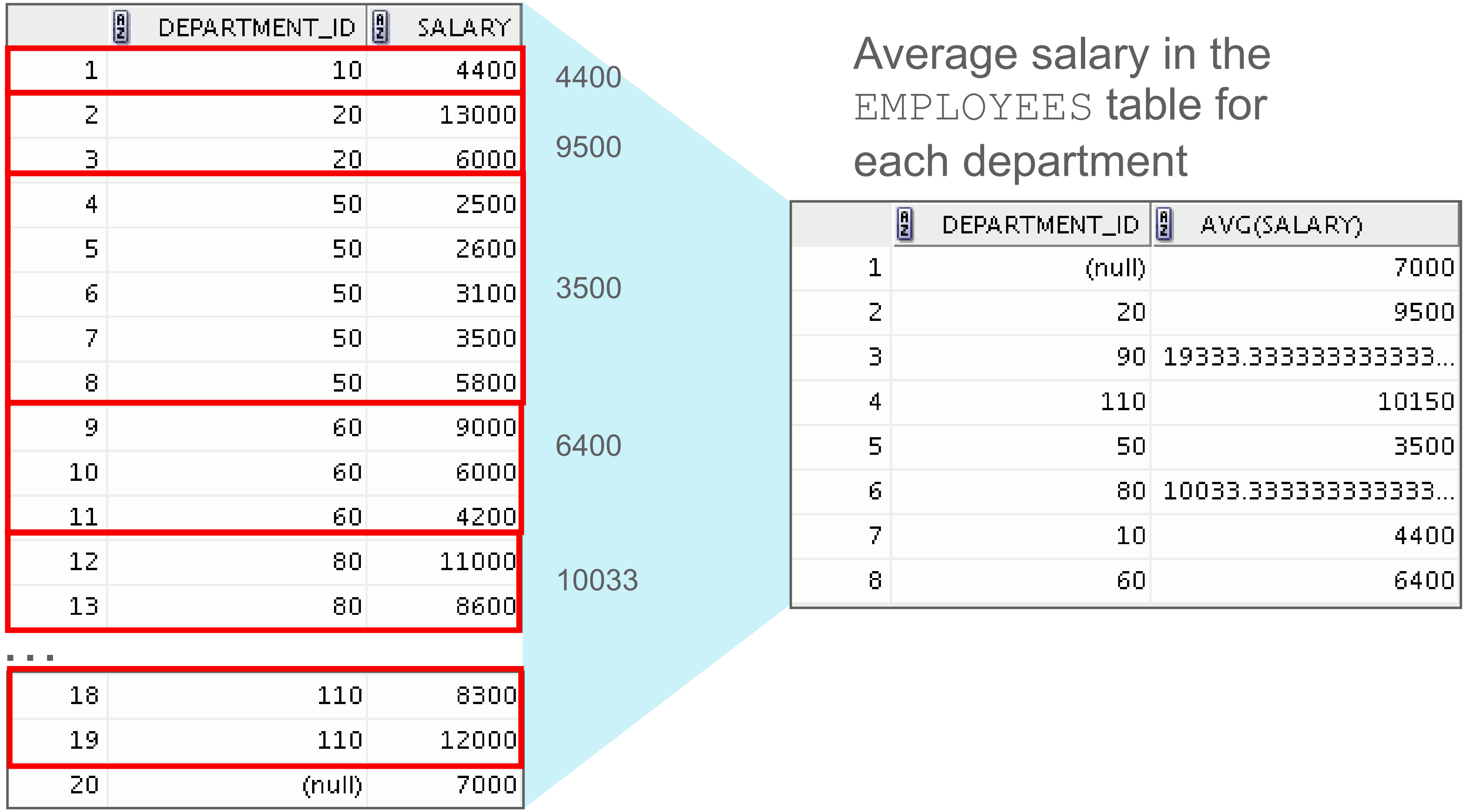
2



	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

Creating Groups of Data

EMPLOYEES



	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

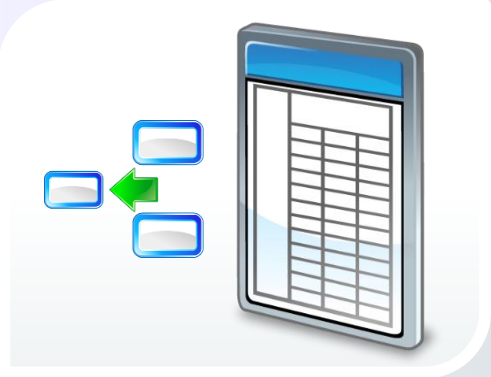
Average salary in the EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.3333333333...
4	110	10150
5	50	3500
6	80	10033.3333333333...
7	10	4400
8	60	6400

Creating Groups of Data: GROUP BY Clause Syntax

You can divide the rows in a table into smaller groups by using the GROUP BY clause.

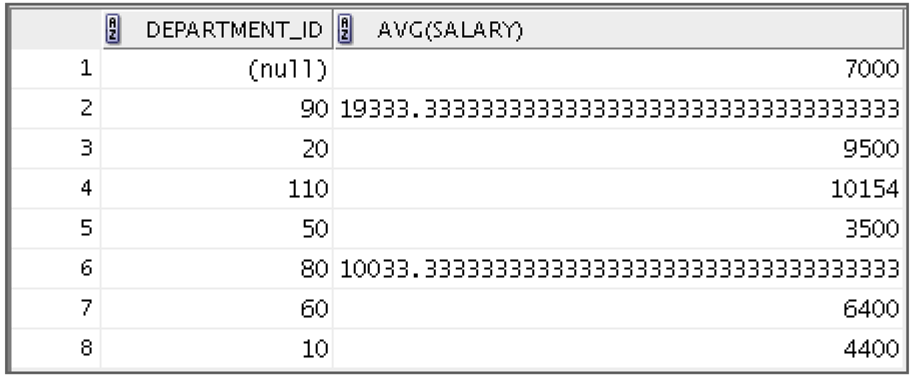
```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```



Using the GROUP BY Clause

All the columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

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

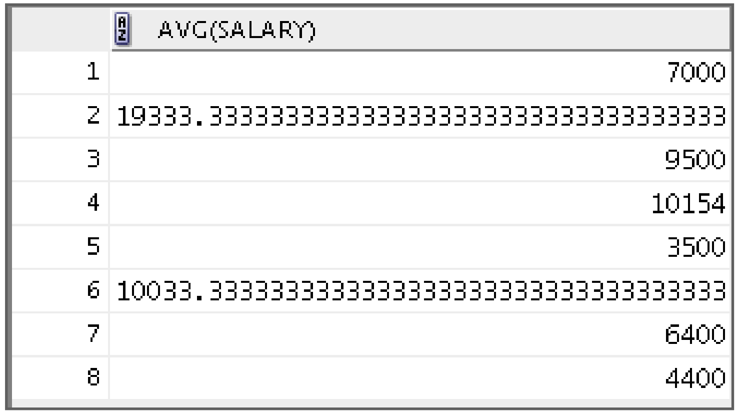


	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.333333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333333
7	60	6400
8	10	4400

Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

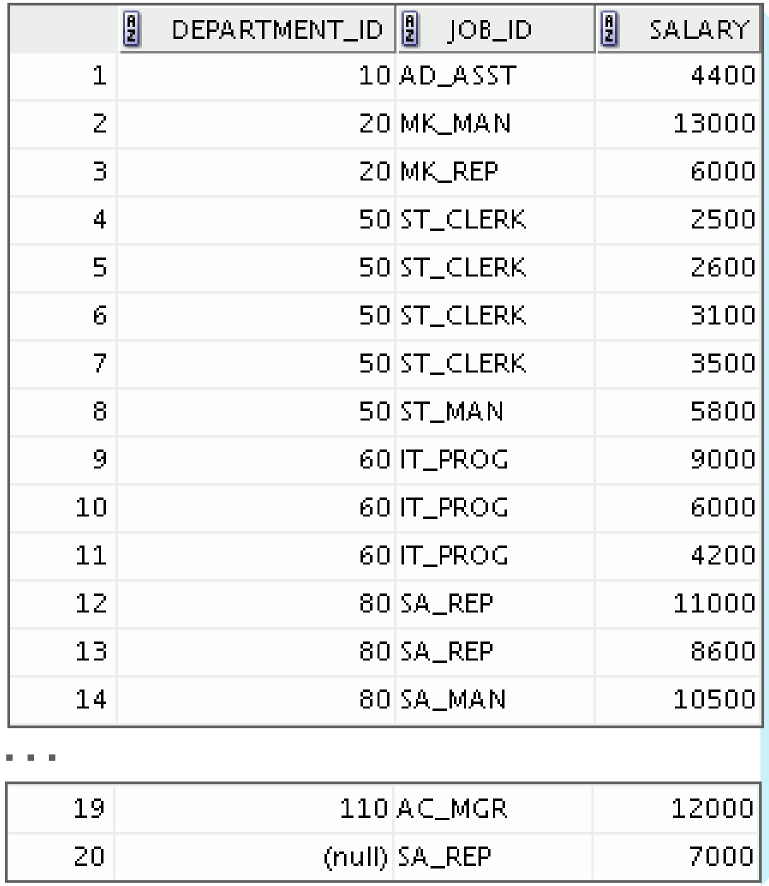
```
SELECT    AVG(salary)
FROM      employees
GROUP BY department_id ;
```



	AVG(SALARY)
1	7000
2	19333.333333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.333333333333333333333333333333333
7	6400
8	4400

Grouping by More Than One Column

EMPLOYEES

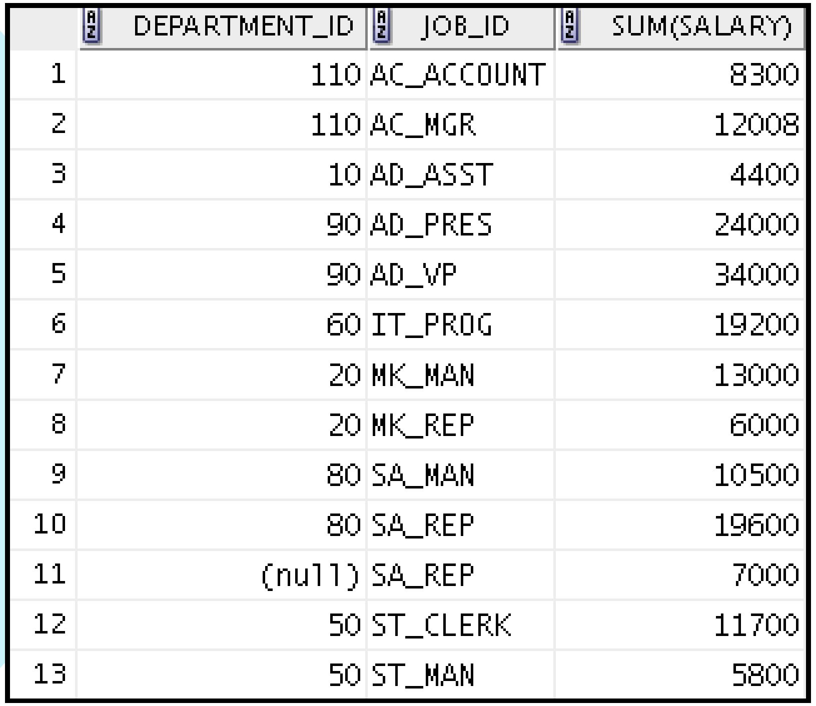


	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500

...

19	110	AC_MGR	12000
20	(null)	SA_REP	7000

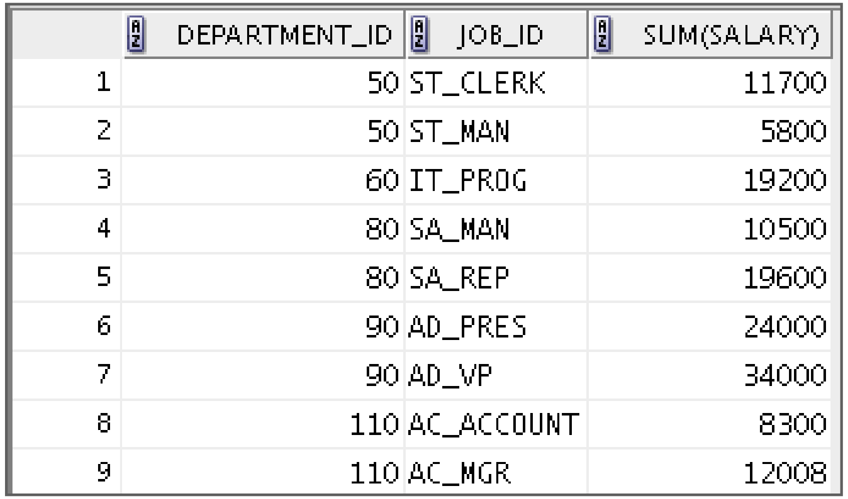
Add the salaries in the EMPLOYEES table for each job, grouped by department.



	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

Using the GROUP BY Clause on Multiple Columns

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE   department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```



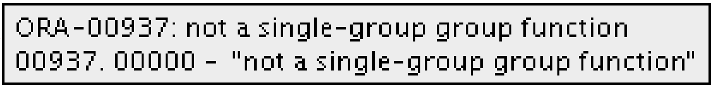
	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

Illegal Queries Using Group Functions

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause:

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

1

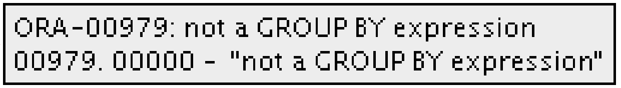


```
ORA-00937: not a single-group group function
00937. 00000 -  "not a single-group group function"
```

A `GROUP BY` clause must be added to count the last names for each `department_id`.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

2



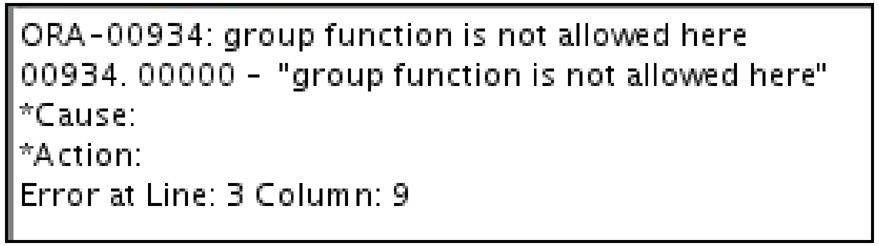
```
ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
```

Either add `job_id` in the `GROUP BY` clause or remove the `job_id` column from the `SELECT` list.

Illegal Queries Using Group Functions

- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

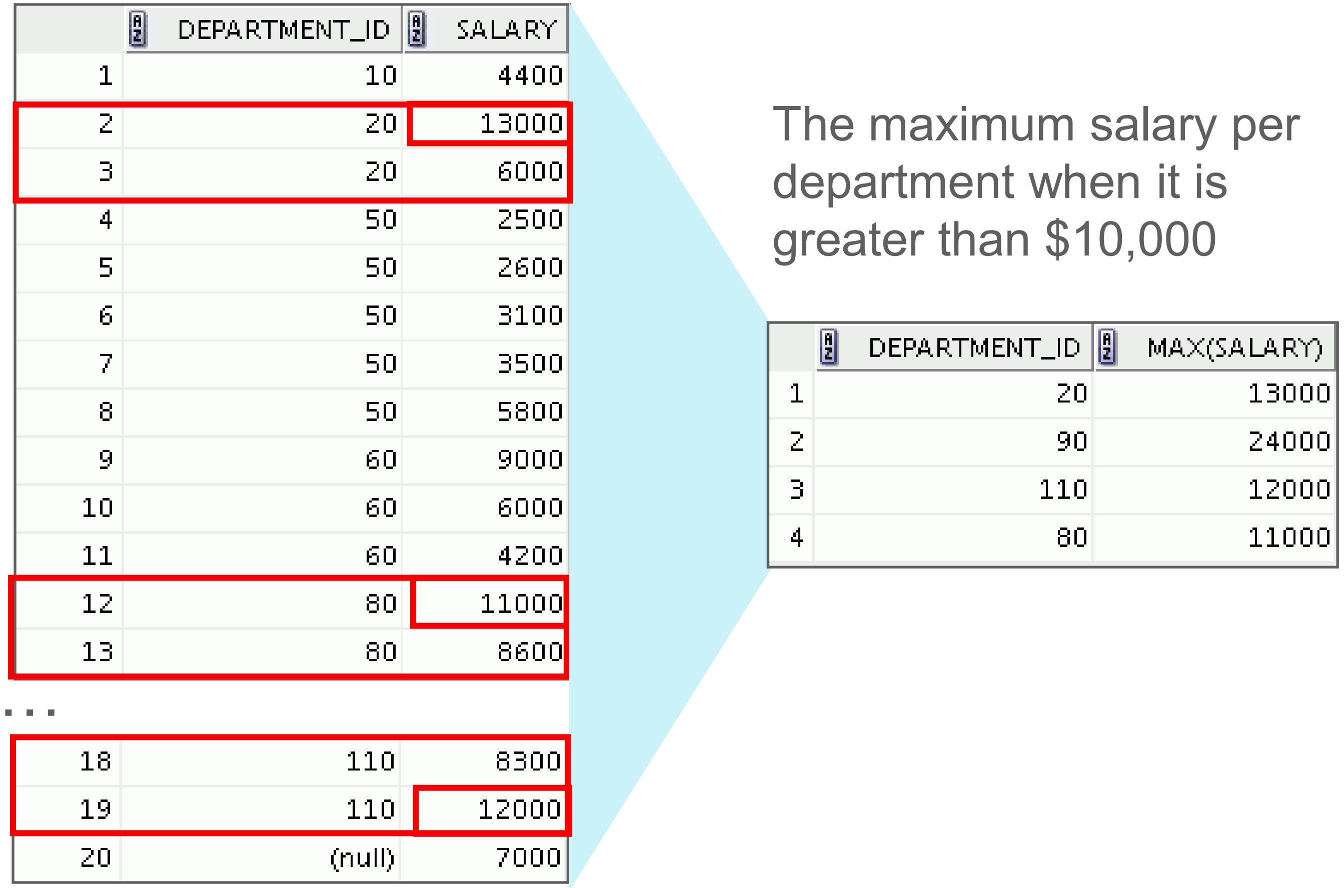


```
ORA-00934: group function is not allowed here
00934. 00000 -  "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

Cannot use the `WHERE` clause to restrict groups

Restricting Group Results

EMPLOYEES



	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	80	8600
…		
18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

Restricting Group Results with the HAVING Clause

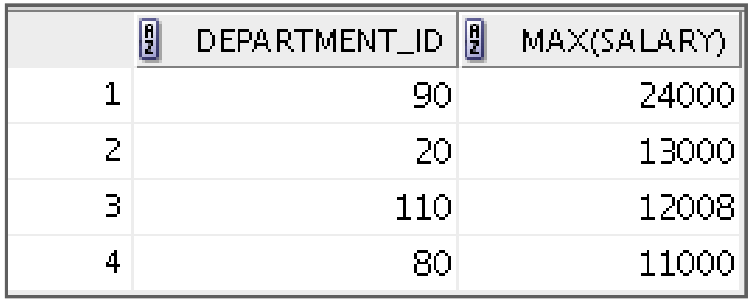
When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY group_by_expression]
[HAVING    group_condition]
[ORDER BY column];
```

Using the HAVING Clause

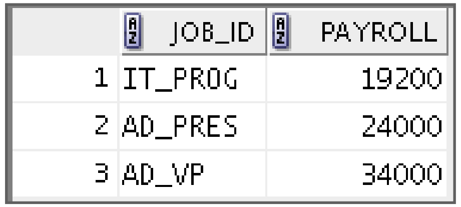
```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)> 10000 ;
```



	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

Using the HAVING Clause

```
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
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

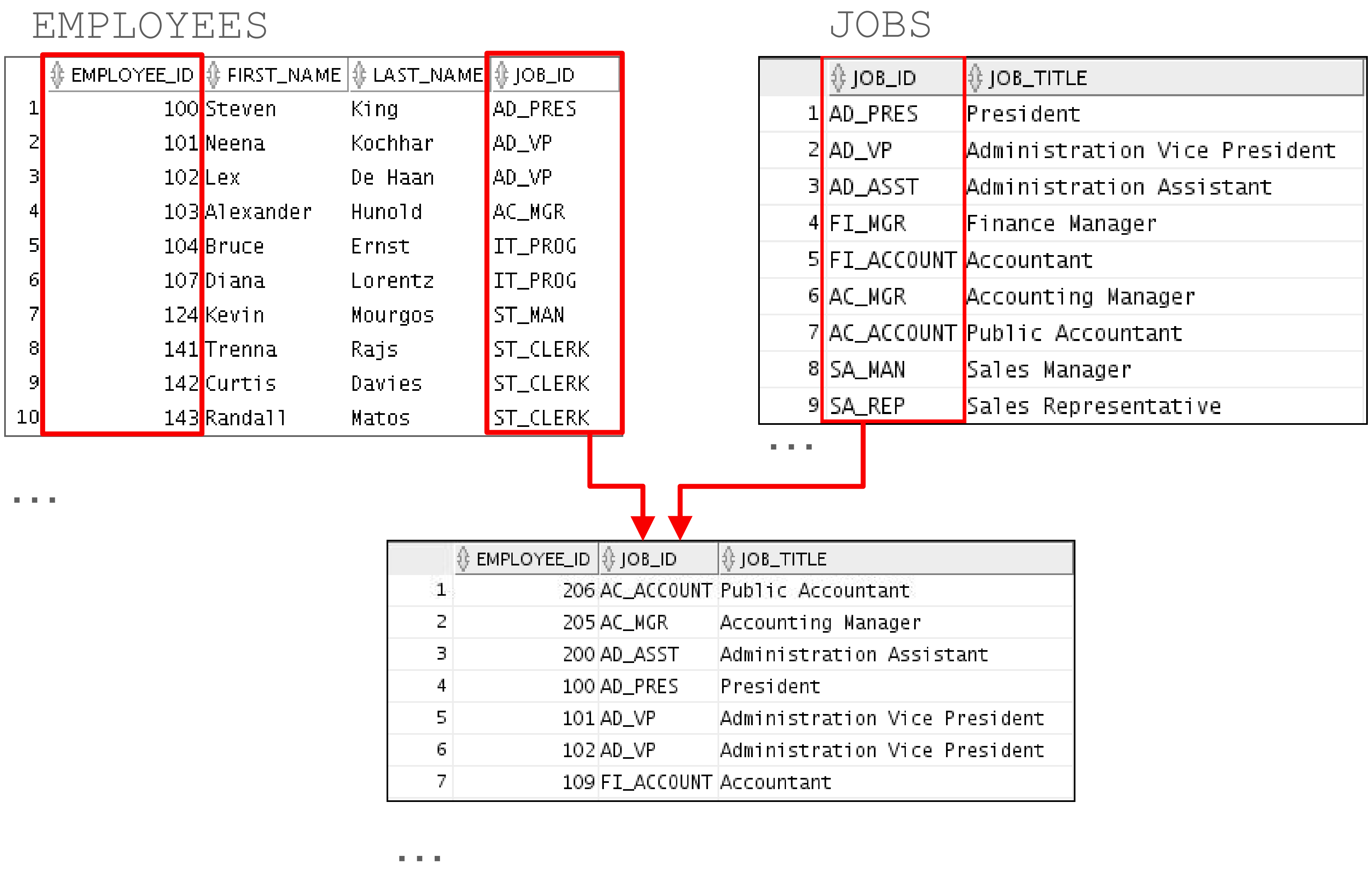
Nesting Group Functions

Display the maximum average salary:

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

Displaying Data from Multiple Tables Using Joins

Obtaining Data from Multiple Tables



EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	AC_MGR
5	104	Bruce	Ernst	IT_PROG
6	107	Diana	Lorentz	IT_PROG
7	124	Kevin	Mourgos	ST_MAN
8	141	Trenna	Rajs	ST_CLERK
9	142	Curtis	Davies	ST_CLERK
10	143	Randall	Matos	ST_CLERK

…

JOBS

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative

…

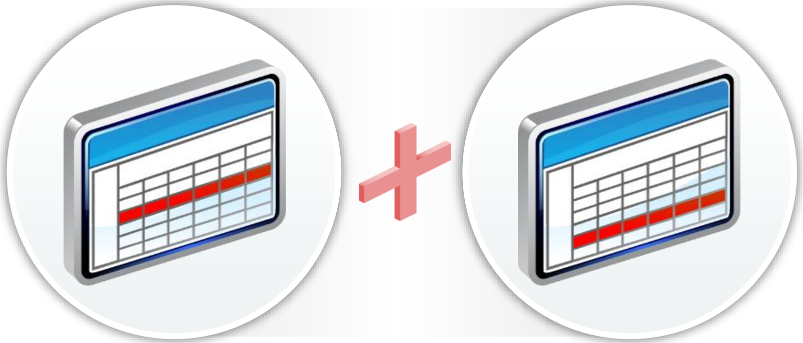
	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

…

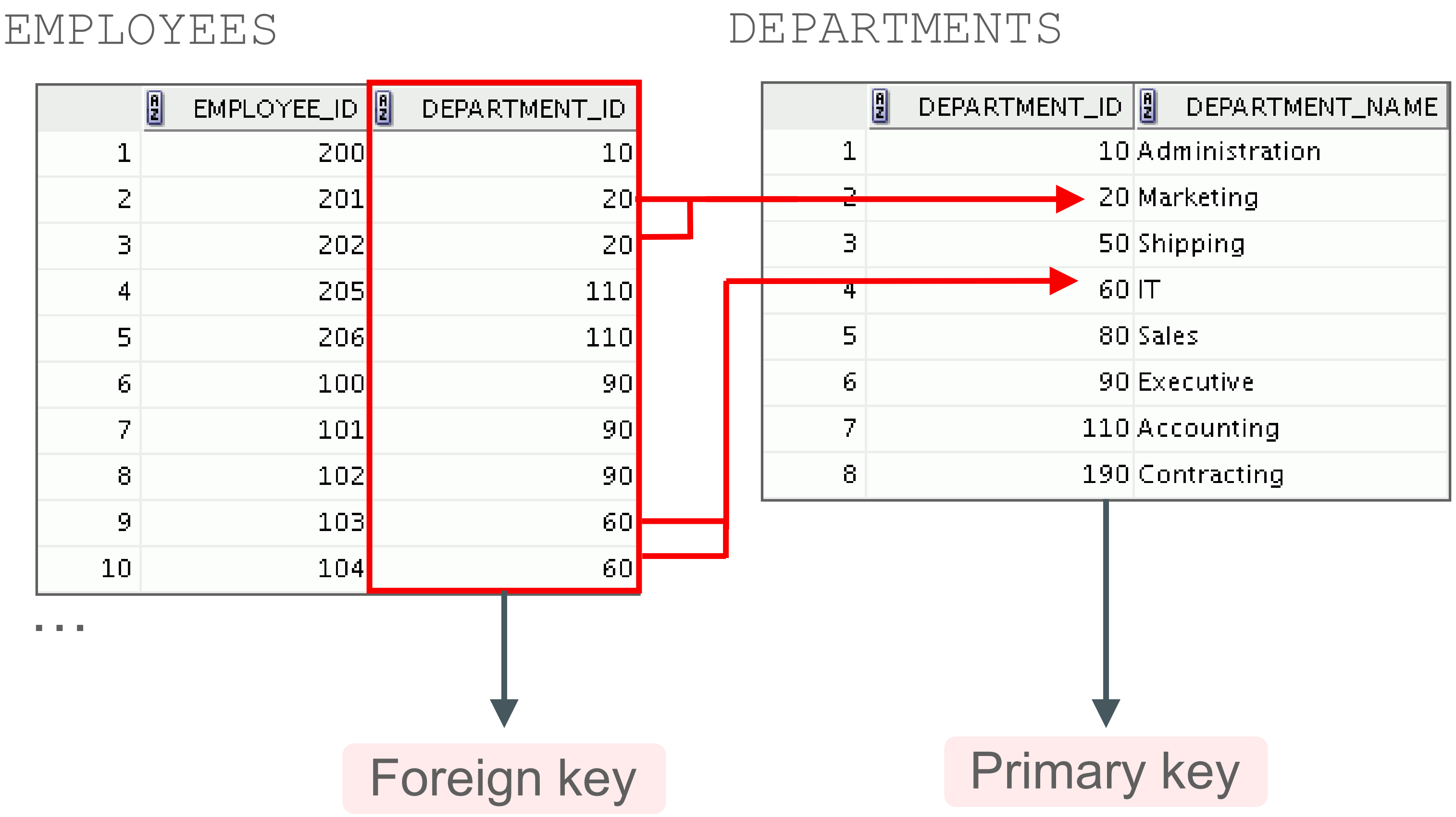
Types of Joins

Joins that are compliant with the ANSI SQL:1999 standard include the following:

- Join with the `ON` clause
- `OUTER` joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross joins



Joining Column Names



EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

…

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Foreign key

Primary key

Qualifying Ambiguous Column Names

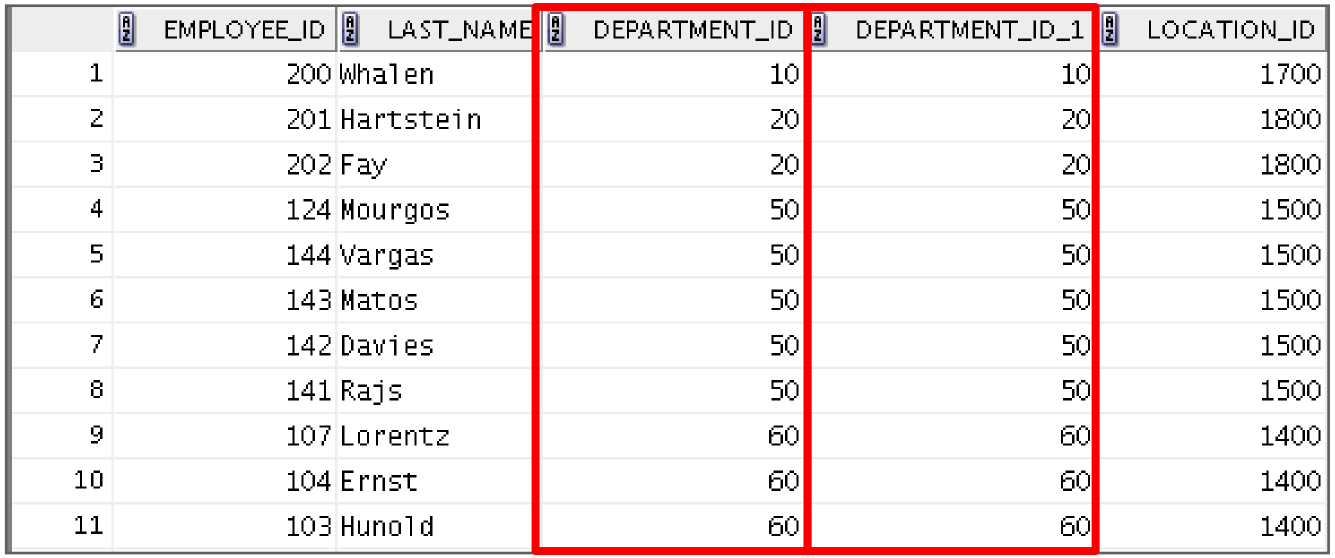
- Use table prefixes to:
  - Qualify column names that are in multiple tables
  - Increase the speed of parsing of a statement
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Creating Joins with the `ON` Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the `ON` clause to specify arbitrary conditions or specify the columns to join.
- Use the `ON` clause to separate the join condition from other search conditions.
- The `ON` clause makes code easy to understand.

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);
```

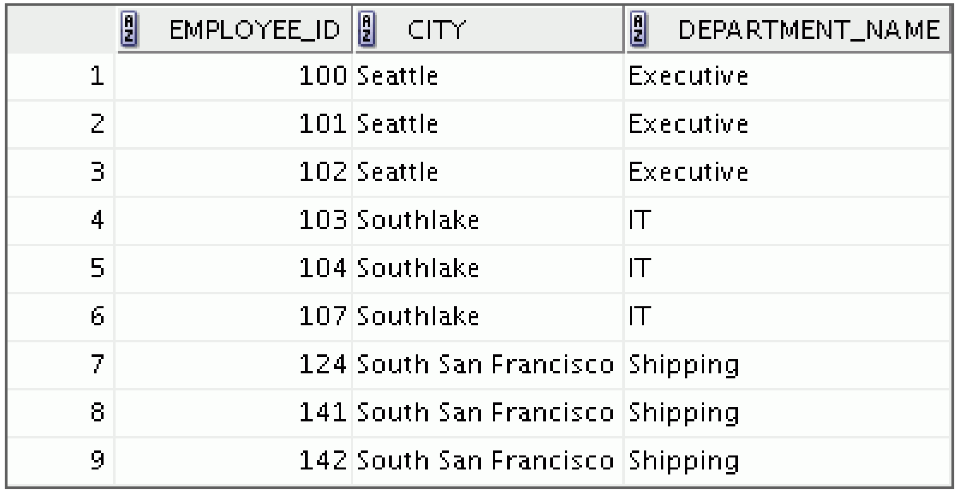


	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400

...

Creating Three-Way Joins

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```



	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

…

Applying Additional Conditions to a Join

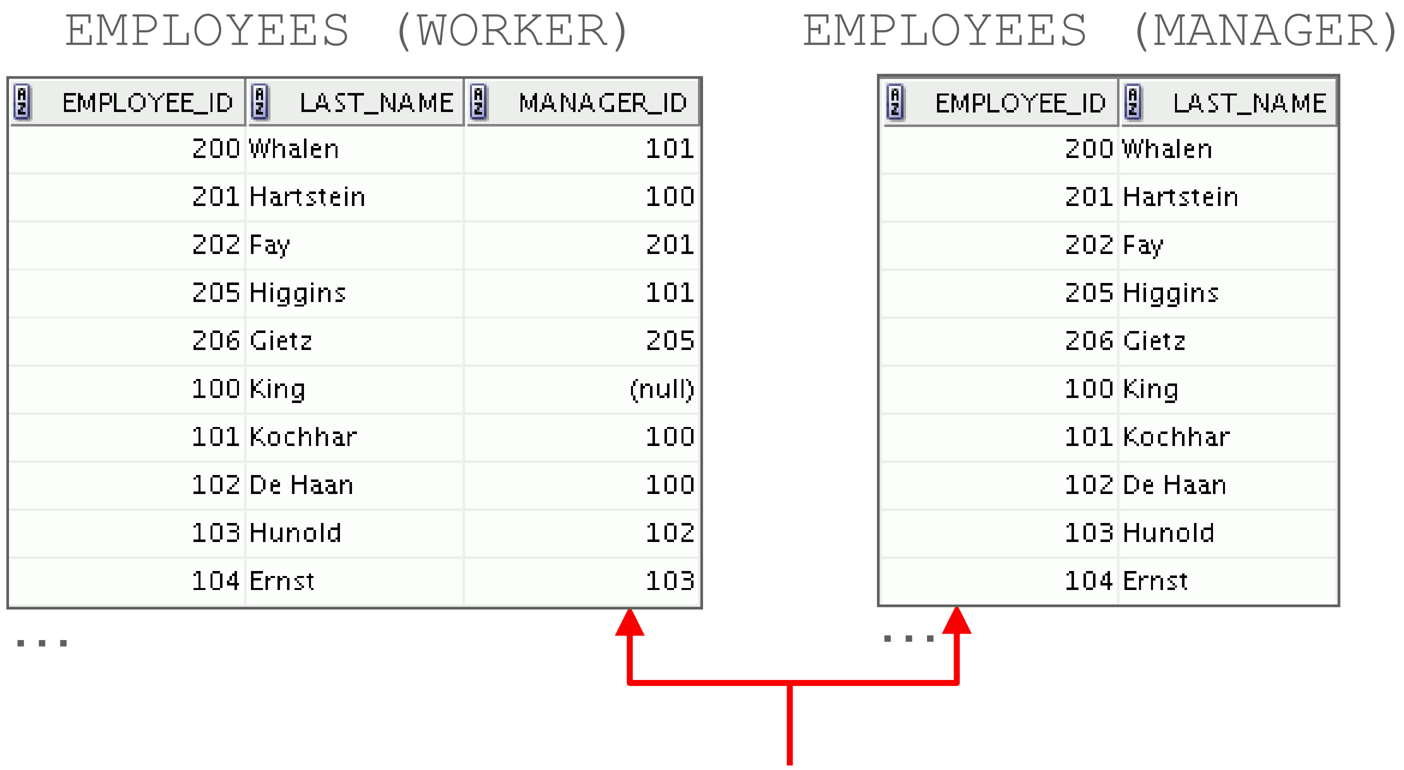
Use the AND clause or the WHERE clause to apply additional conditions:

```
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)
AND    e.manager_id = 149 ;
```

OR

```
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)
WHERE   e.manager_id = 149 ;
```

Joining a Table to Itself



EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

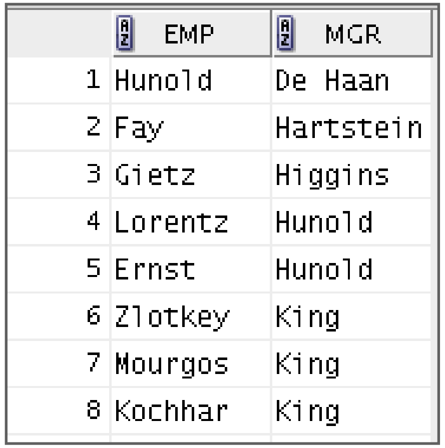
EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

Self-Joins Using the ON Clause

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

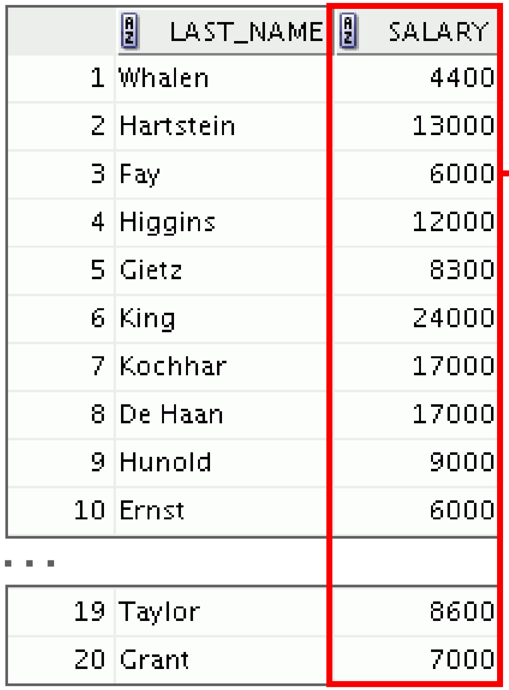


	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King

...

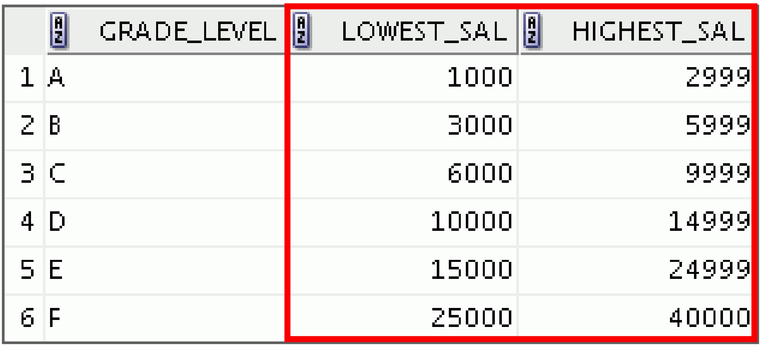
Nonequijoins

EMPLOYEES



	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES



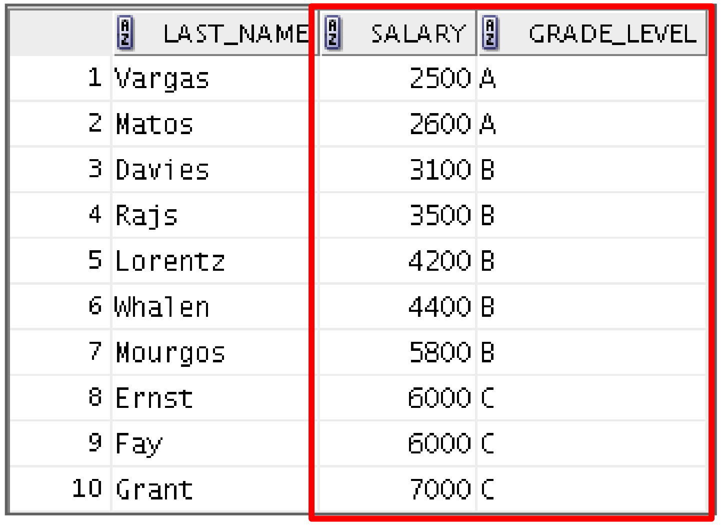
	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

The JOB_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL.

Therefore, the GRADE_LEVEL column can be used to assign grades to each employee based on his salary.

Retrieving Records with Nonequijoins

```
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;
```

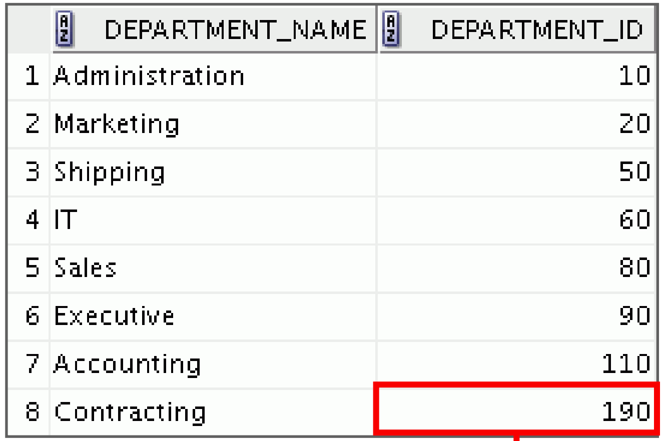


	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

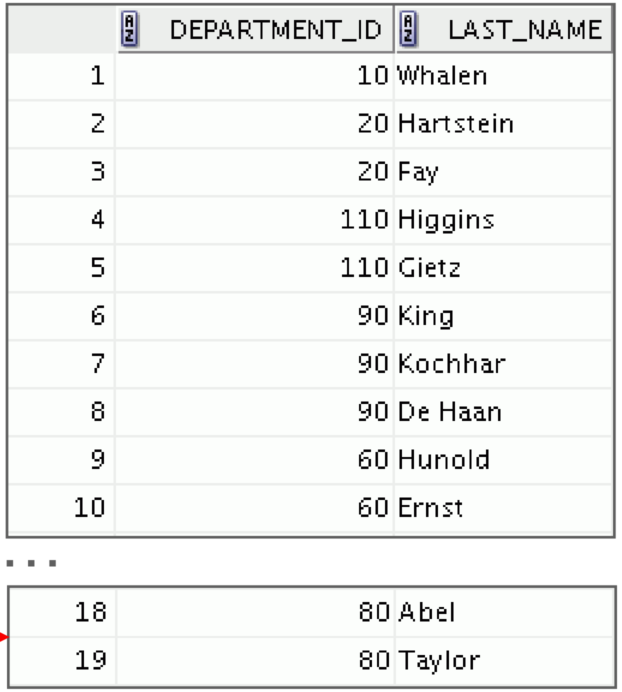
Returning Records with No Direct Match Using `OUTER` Joins

`DEPARTMENTS`



	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

Equijoin with `EMPLOYEES`



	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst
…		
18	80	Abel
19	80	Taylor

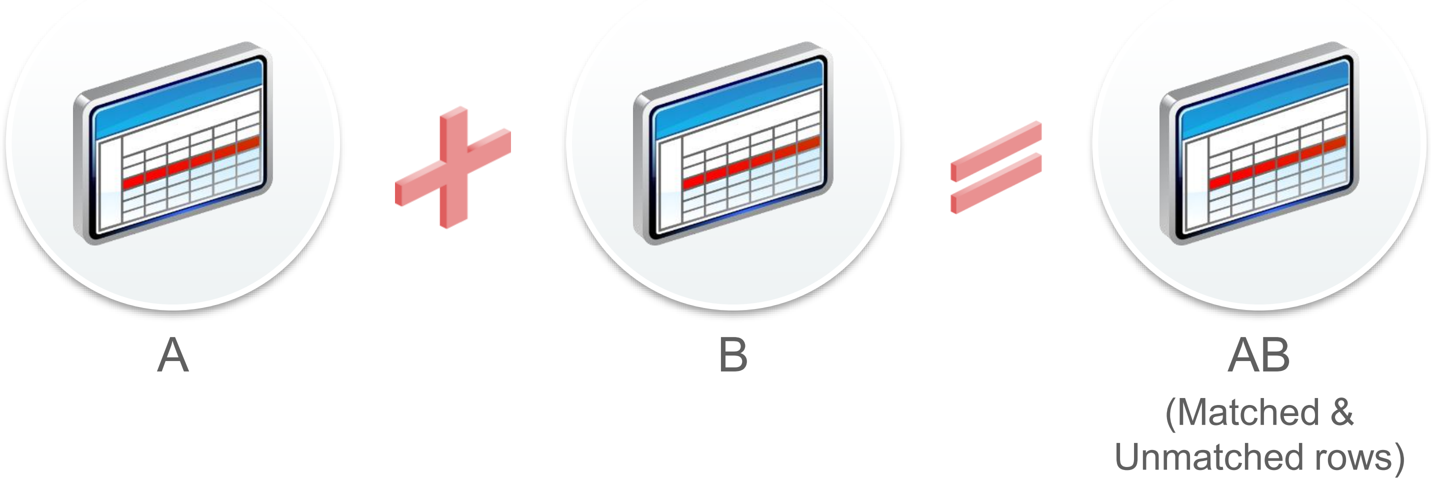
There are no employees in department **`190`**.

Employee "**`Grant`**" has not been assigned a department ID.

Therefore, the above two records do not appear in the equijoin result.

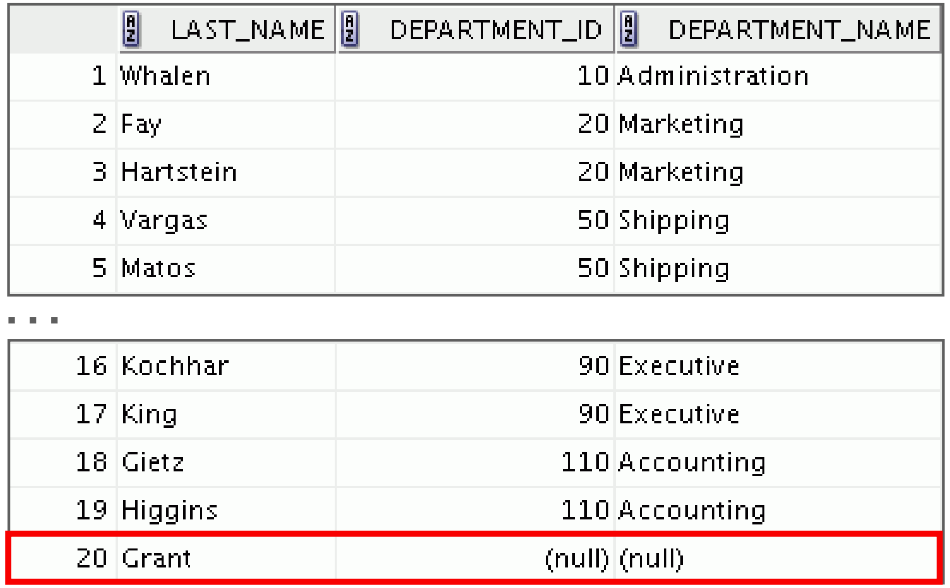
INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an INNER join.
- A join between two tables that returns the results of the INNER join as well as the unmatched rows from the left (or right) table is called a LEFT (or RIGHT) OUTER join.
- A join between two tables that returns the results of an INNER join as well as the results of a left and right join is a FULL OUTER JOIN.



LEFT 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) ;
```



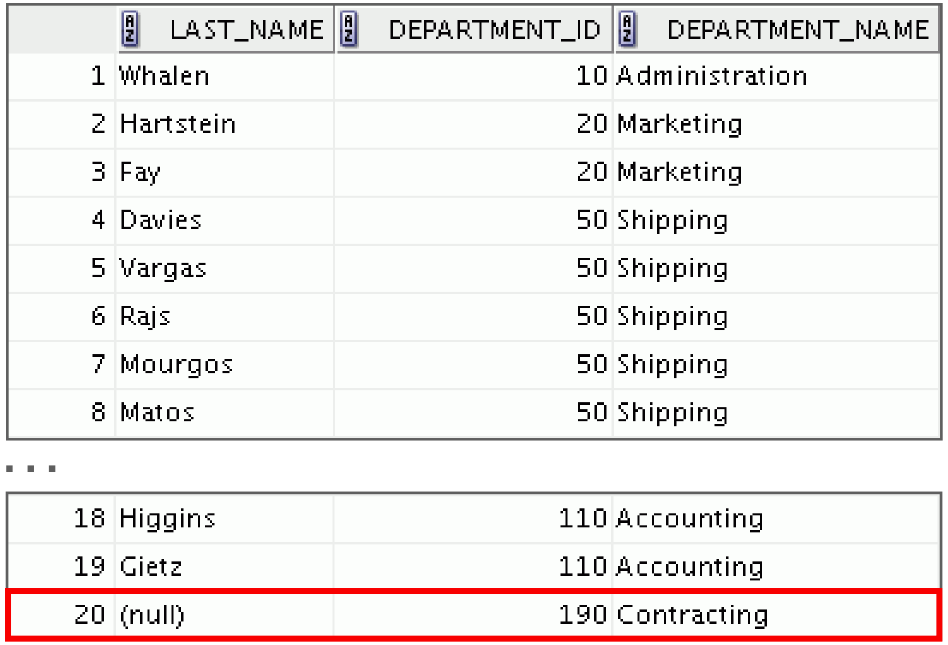
	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

. . .

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

RIGHT OUTER JOIN

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



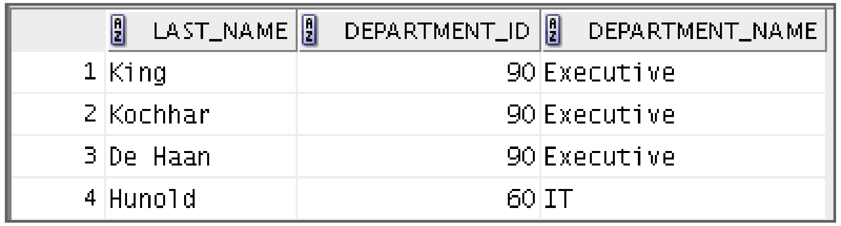
	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

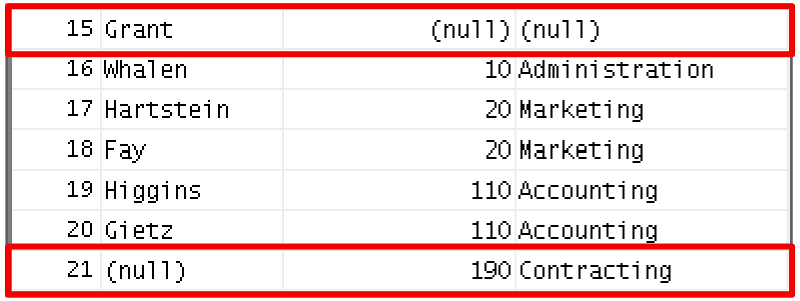
FULL OUTER JOIN

```
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) ;
```



	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

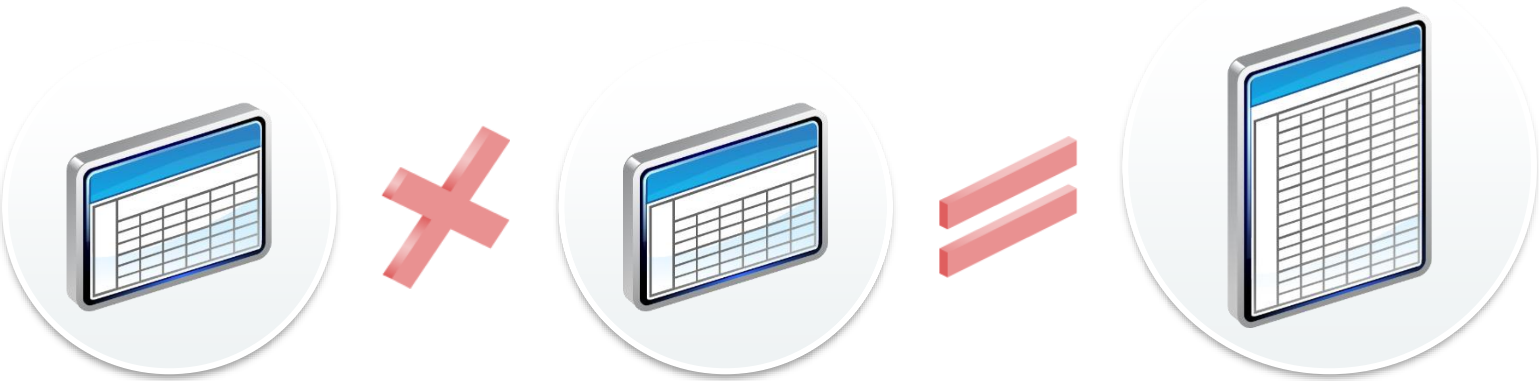


15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

Cartesian Products

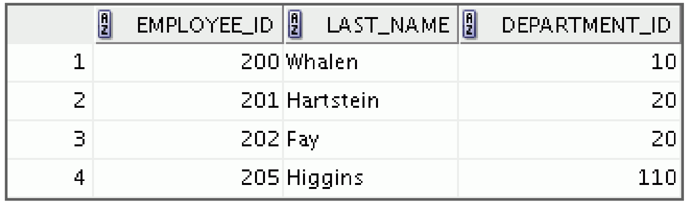
A Cartesian product:

- Is a join of every row of one table to every row of another table
- Generates a large number of rows and the result is rarely useful



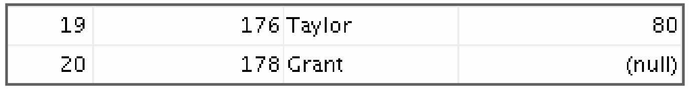
Generating a Cartesian Product

EMPLOYEES (20 rows)



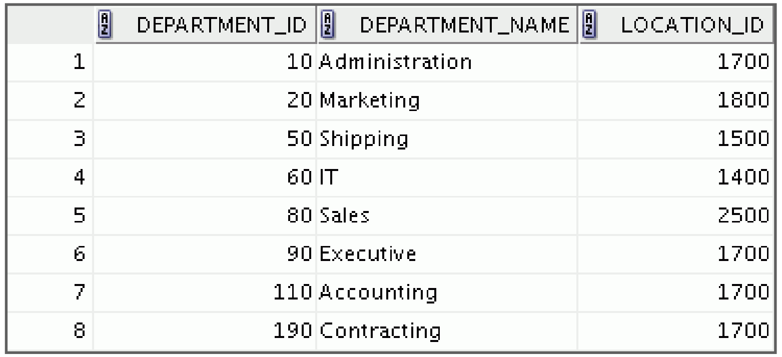
	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110

...



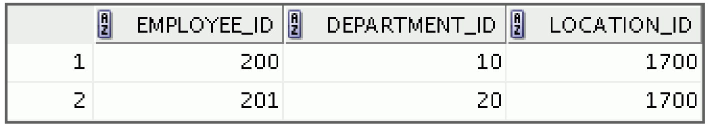
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS (8 rows)



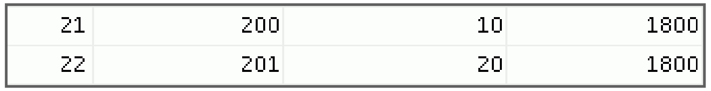
	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product:
20 x 8 = 160 rows



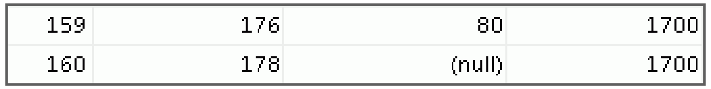
	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700

...



21	200	10	1800
22	201	20	1800

...

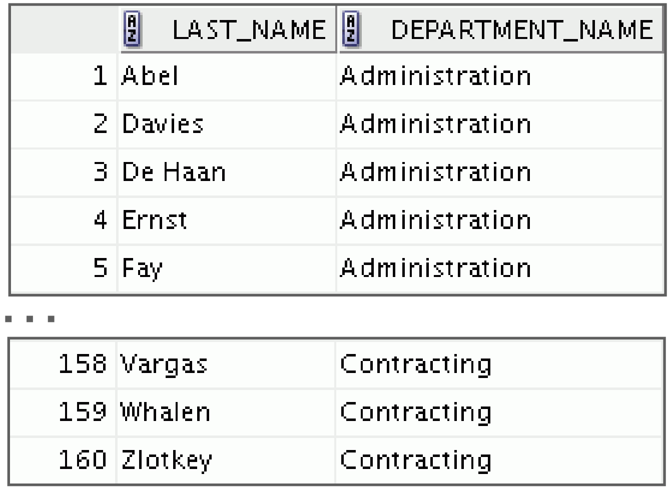


159	176	80	1700
160	178	(null)	1700

Creating Cross Joins

- A CROSS JOIN is a JOIN operation that produces a Cartesian product of two tables.
- To create a Cartesian product, specify CROSS JOIN in your SELECT statement.

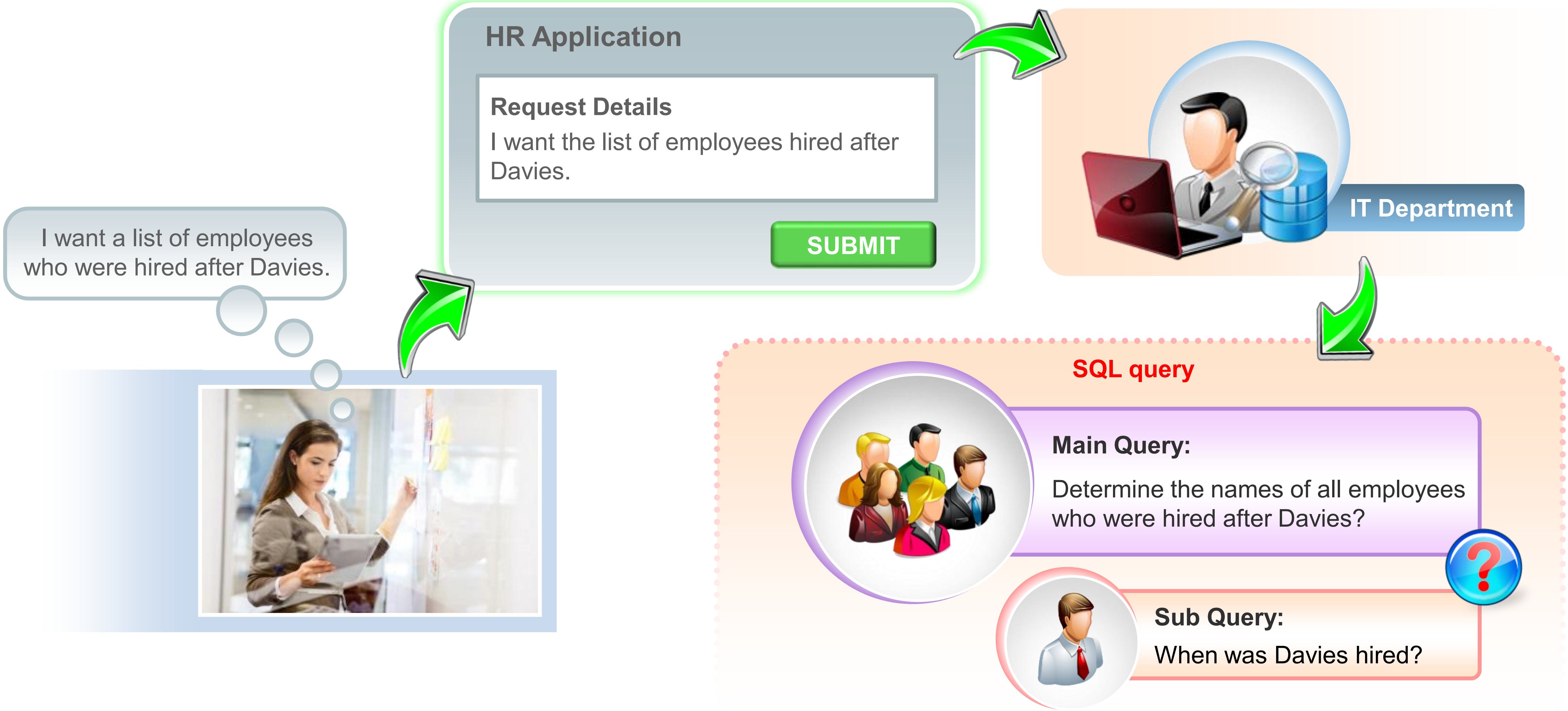
```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```



	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

Using Subqueries to Solve Queries

Using a Subquery to Solve a Problem



I want a list of employees who were hired after Davies.

HR Application

Request Details

I want the list of employees hired after Davies.

SUBMIT

IT Department

SQL query

Main Query:

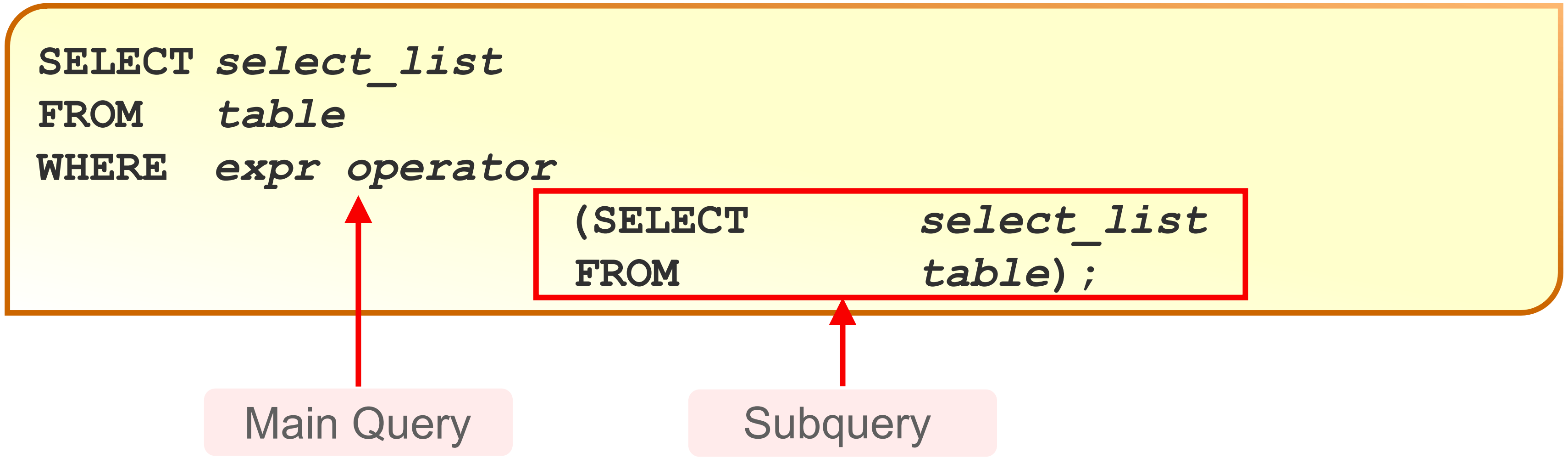
Determine the names of all employees who were hired after Davies?

Sub Query:

When was Davies hired?

Subquery Syntax

- The subquery (inner query) executes *before* the main query (outer query).
- The result of the subquery is used by the main query.

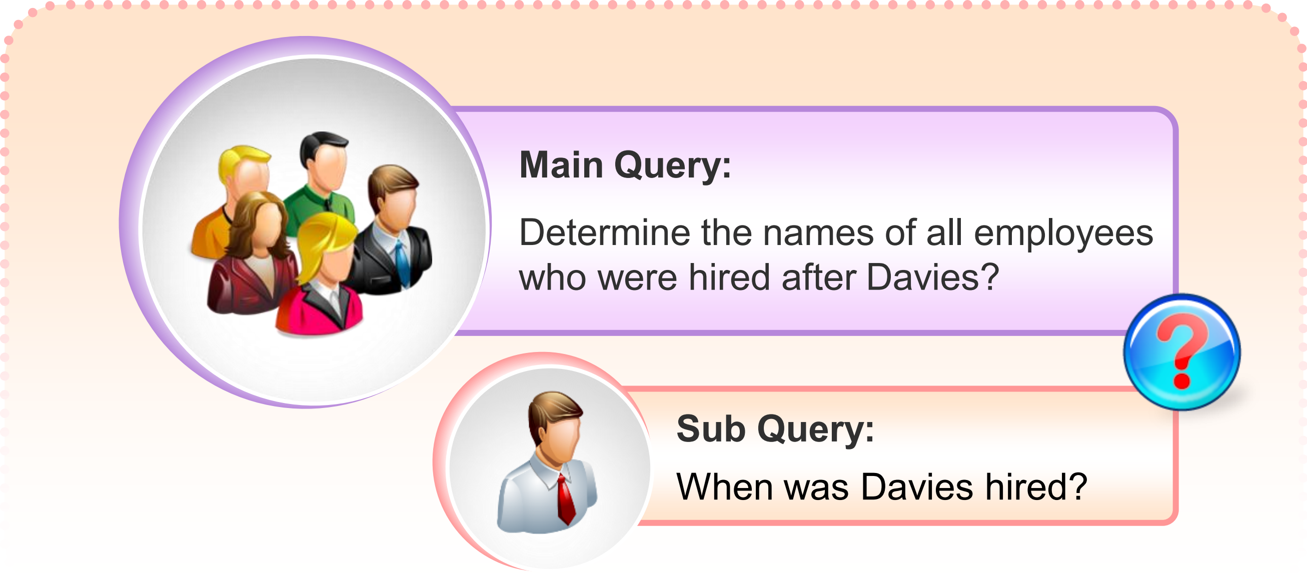


```
SELECT select_list
FROM   table
WHERE  expr operator
                     (SELECT        select_list
                      FROM          table);
```

Main Query

Subquery

Using a Subquery



Main Query:

Determine the names of all employees who were hired after Davies?

Sub Query:

When was Davies hired?

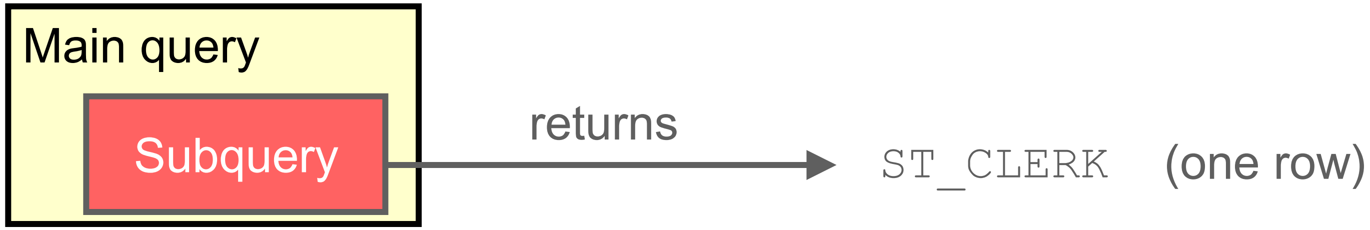
```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date > (SELECT hire_date
                    FROM   employees
                    WHERE  last_name = 'Davies');
```

Rules and Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability. (However, the subquery can appear on either side of the comparison operator.)
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

Types of Subqueries

- Single-row subquery



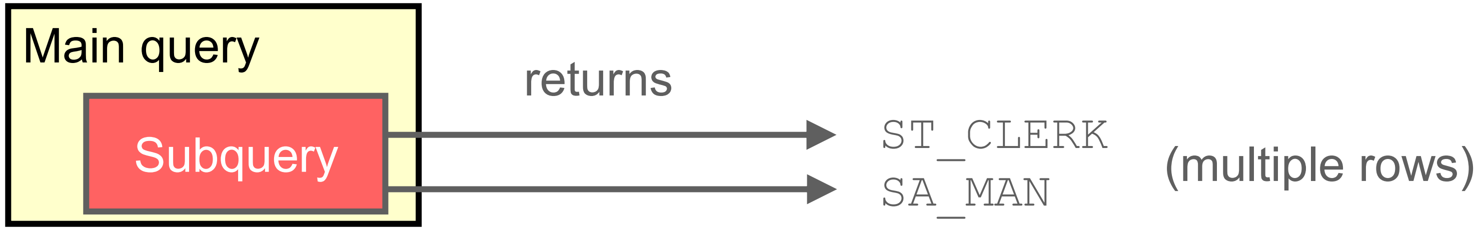
Main query

Subquery

returns

ST_CLERK (one row)

- Multiple-row subquery



Main query

Subquery

returns

ST_CLERK

SA_MAN

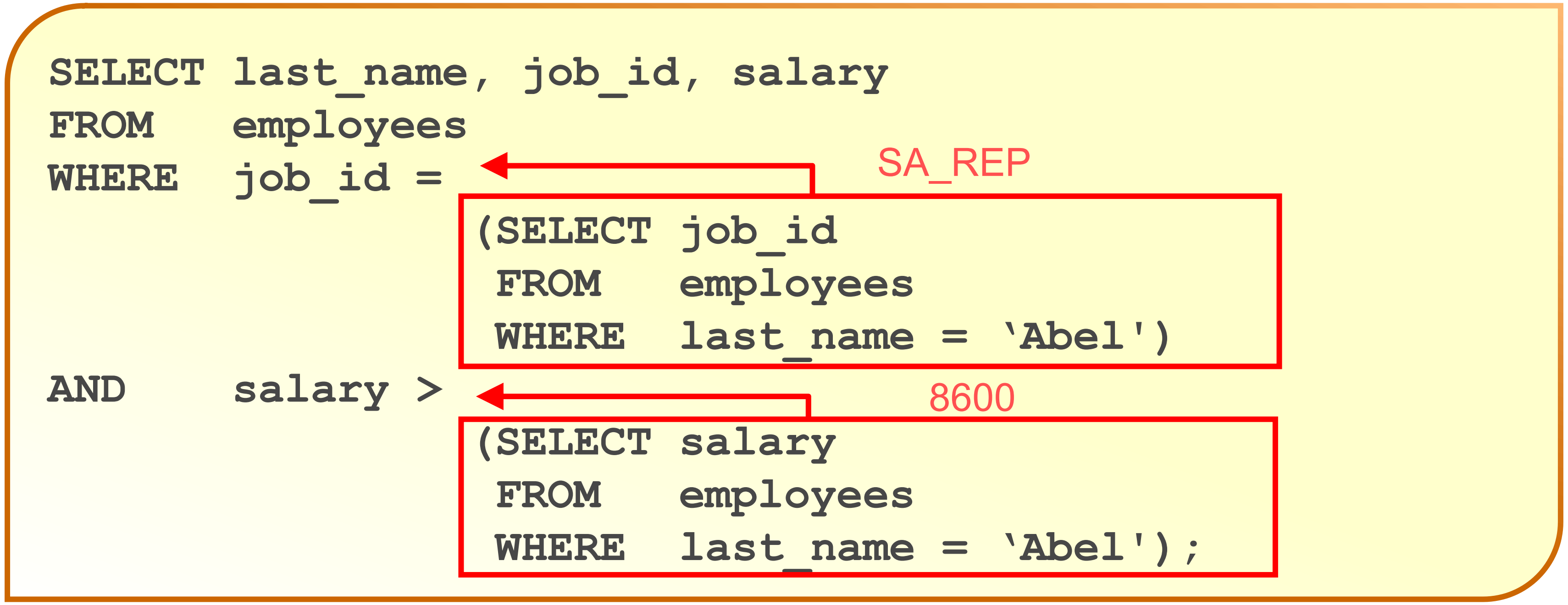
(multiple rows)

Single-Row Subqueries

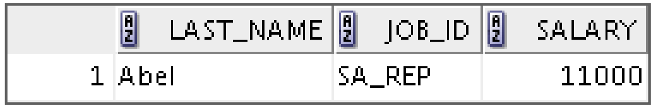
- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Executing Single-Row Subqueries

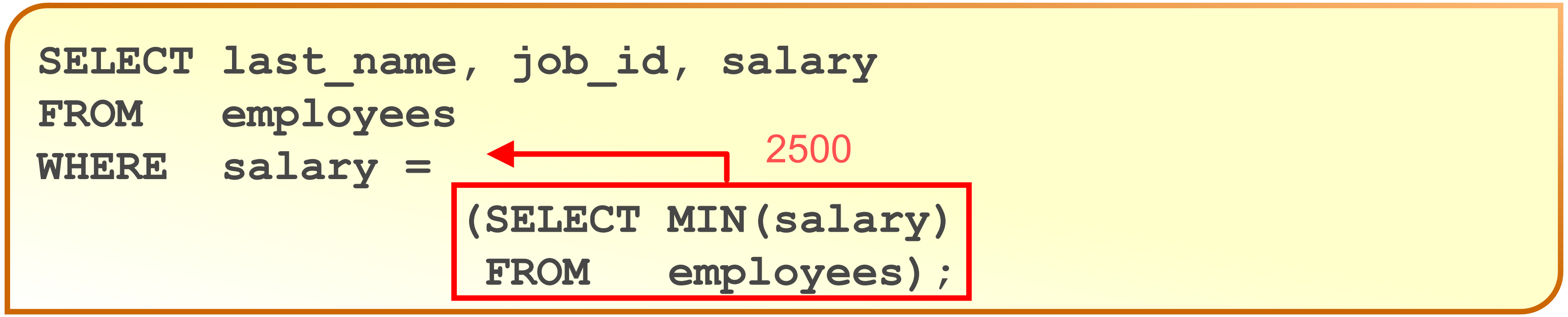


```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = ‘Abel')
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = ‘Abel');
```



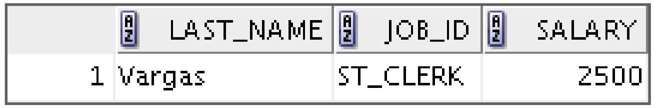
	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

Using Group Functions in a Subquery



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

2500

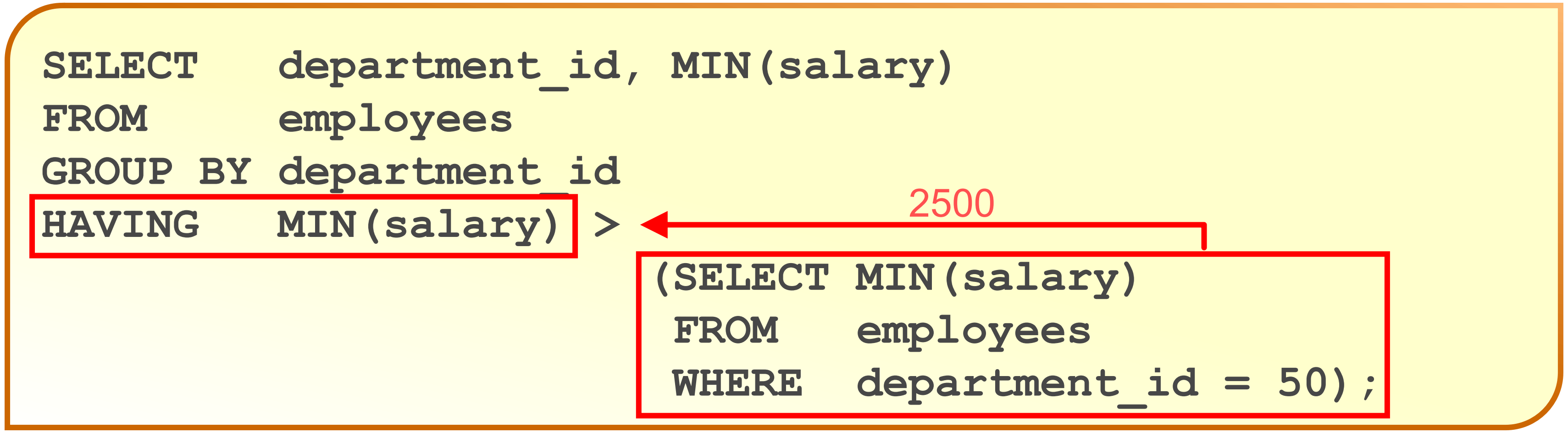


	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

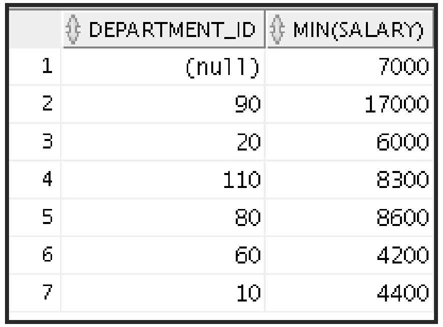
HAVING Clause with Subqueries

The Oracle server:

- Executes the subqueries first
- Returns the result into the HAVING clause of the main query



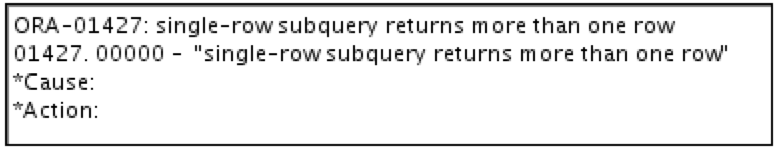
```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                       (SELECT MIN(salary)
                        FROM   employees
                        WHERE  department_id = 50);
```



	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
4	110	8300
5	80	8600
6	60	4200
7	10	4400

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
```

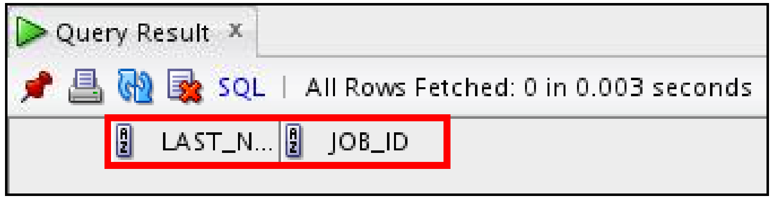


```
ORA-01427: single-row subquery returns more than one row
01427. 00000 -  "single-row subquery returns more than one row"
*Cause:
*Action:
```

Single-row operator with multiple-row subquery

No Rows Returned by the Inner Query

```
SELECT last_name, job_id
FROM   employees
WHERE job_id =
                 (SELECT job_id
                  FROM   jobs
                  WHERE  job_title = ‘Architect');
```



Query Result

SQL | All Rows Fetched: 0 in 0.003 seconds

```
LAST_N... JOB_ID
```

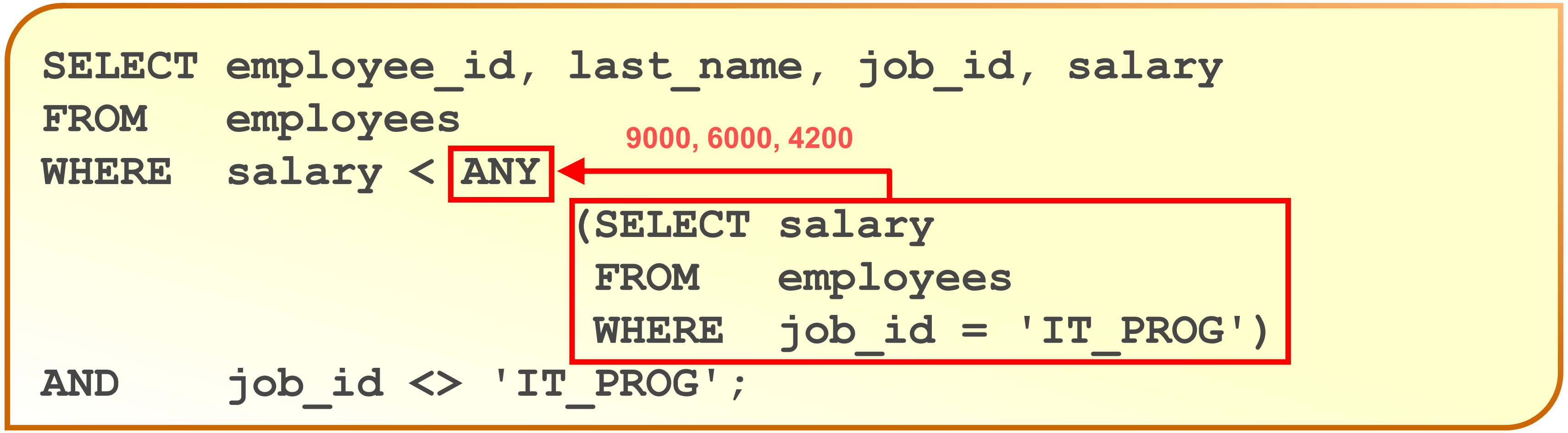
The subquery returns no rows because there is no job with the title “Architect.”

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

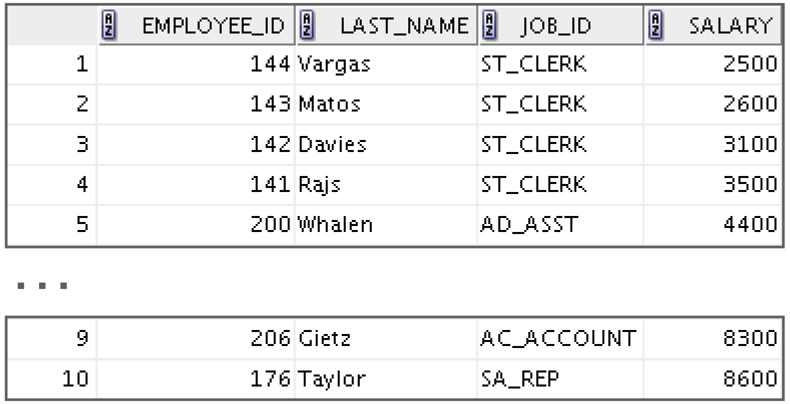
Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if at least one element exists in the result set of the subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if the relation is TRUE for all elements in the result set of the subquery.

Using the ANY Operator in Multiple-Row Subqueries



```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

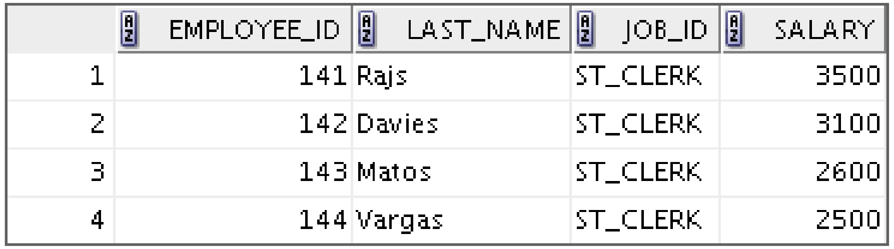


	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

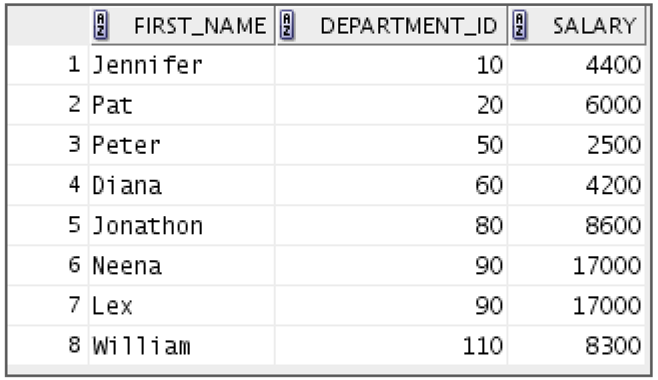


	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

Multiple-Column Subquery: Example

Display all the employees with the lowest salary in each department.

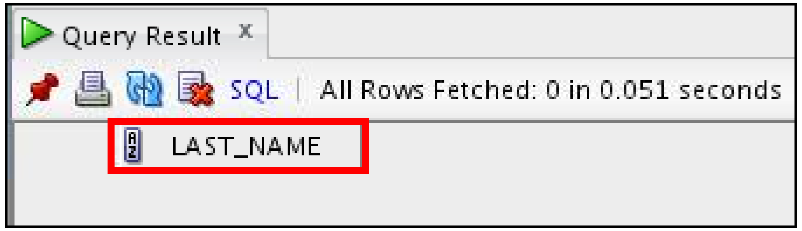
```
SELECT first_name, department_id, salary
FROM employees
WHERE (salary) IN
      (SELECT min(salary)
       FROM employees)
ORDER BY department_id;
```



	FIRST_NAME	DEPARTMENT_ID	SALARY
1	Jennifer	10	4400
2	Pat	20	6000
3	Peter	50	2500
4	Diana	60	4200
5	Jonathon	80	8600
6	Neena	90	17000
7	Lex	90	17000
8	William	110	8300

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);
```



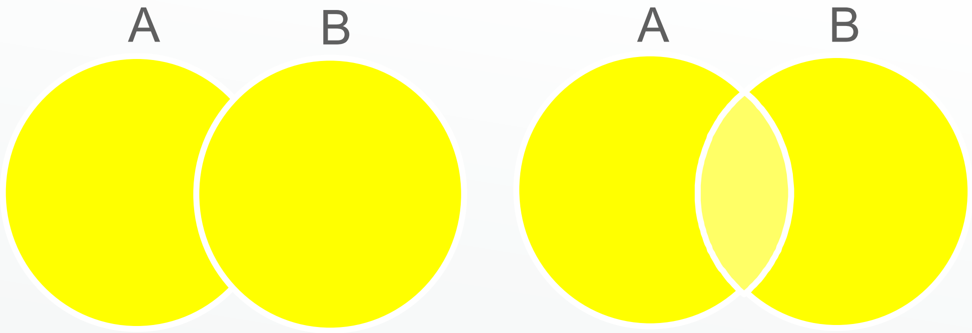
All Rows Fetched: 0 in 0.051 seconds

LAST_NAME

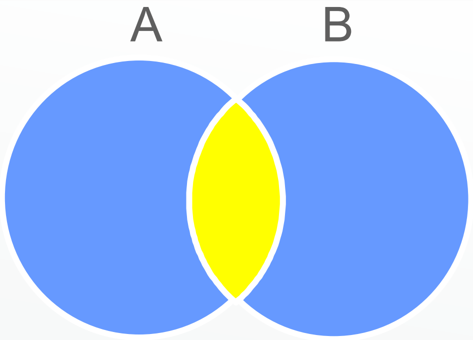
The subquery returns no rows because one of the values returned by a subquery is null.

Using Set Operators

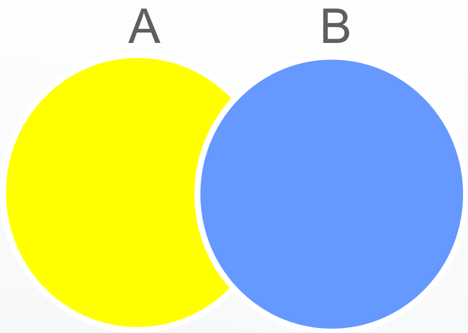
Set Operators



UNION/UNION ALL



INTERSECT

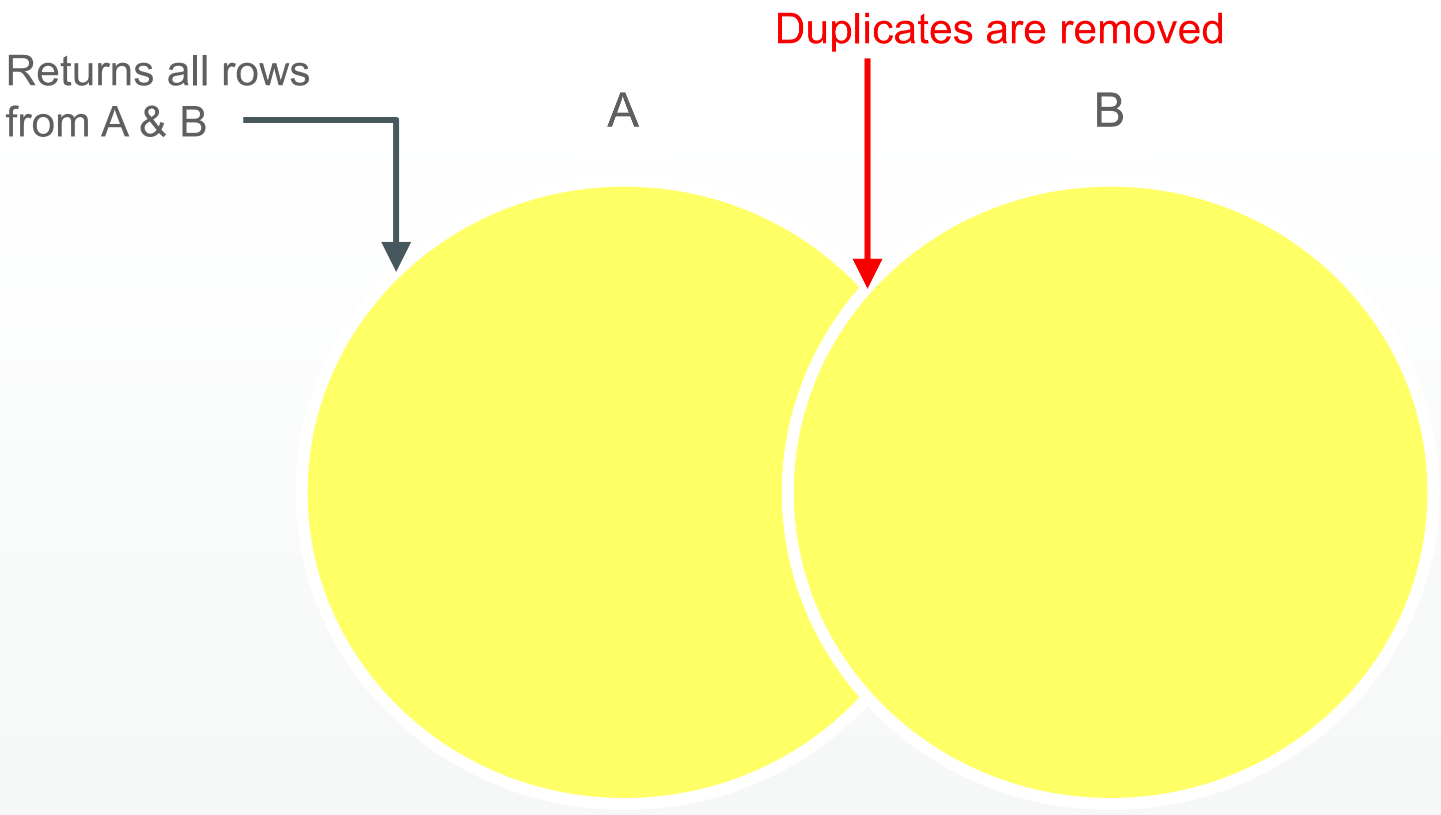


EXCEPT

Set Operator Rules

- The expressions in the `SELECT` lists must match in number.
- The data type of each column in the subsequent query must match the data type of its corresponding column in the first query.
- Parentheses can be used to alter the sequence of execution.
- The `ORDER BY` clause can appear only at the very end of the statement.

UNION Operator



Duplicates are removed

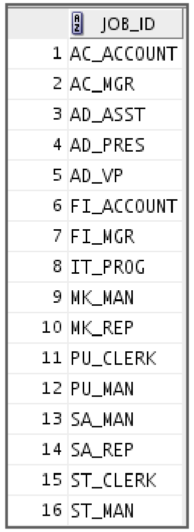
Returns all rows from A & B

The UNION operator returns rows from both queries after eliminating duplications.

Using the UNION Operator

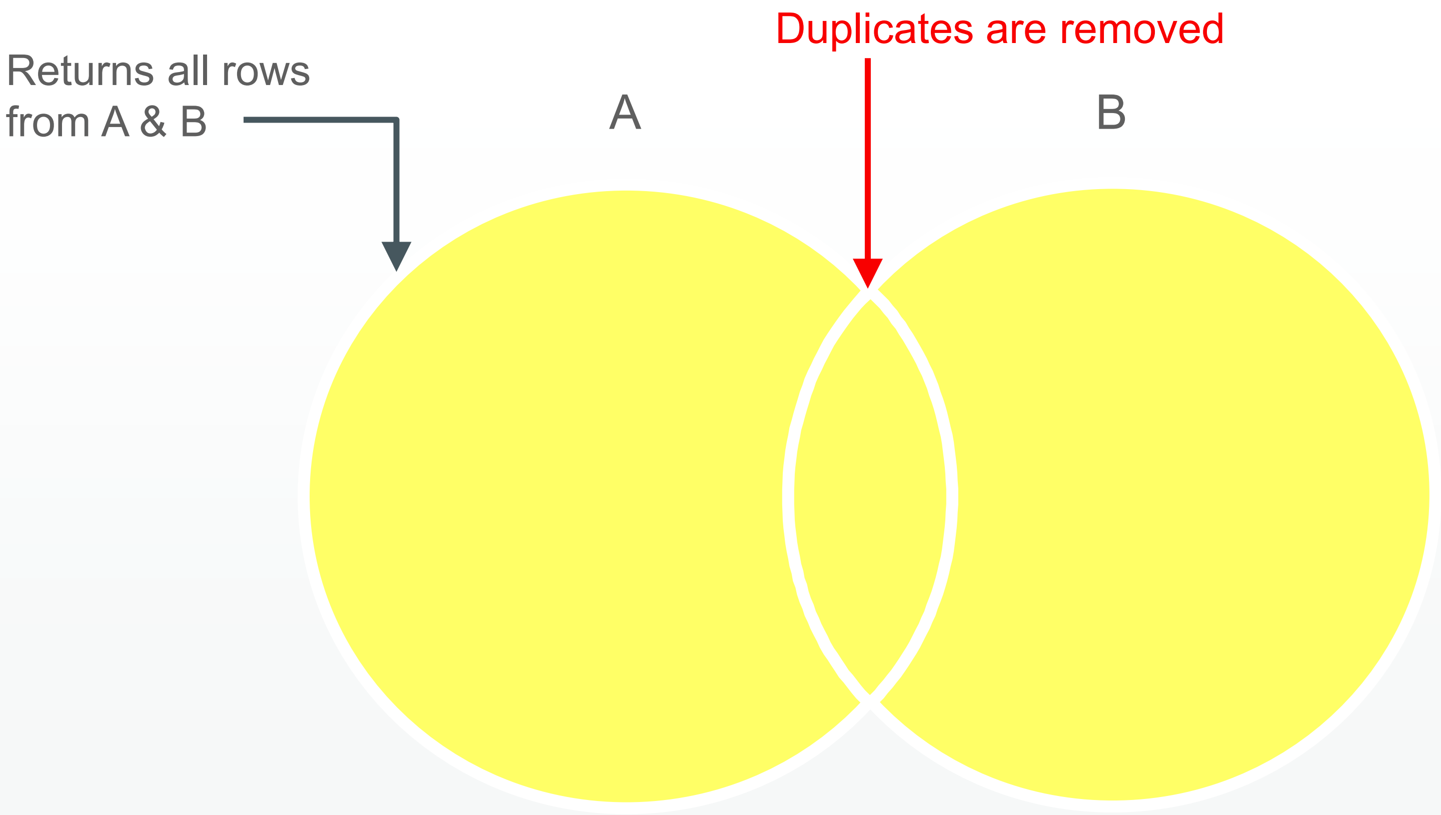
Display the job details of all the current and retired employees. Display each job only once.

```
SELECT job_id
FROM   employees
UNION
SELECT job_id
FROM job_history
```



	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	FI_ACCOUNT
7	FI_MGR
8	IT_PROG
9	MK_MAN
10	MK_REP
11	PU_CLERK
12	PU_MAN
13	SA_MAN
14	SA_REP
15	ST_CLERK
16	ST_MAN

UNION ALL Operator

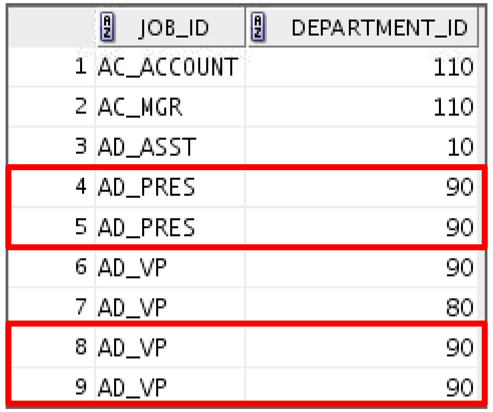


The UNION ALL operator returns rows from both queries, including all duplications.

Using the `UNION ALL` Operator

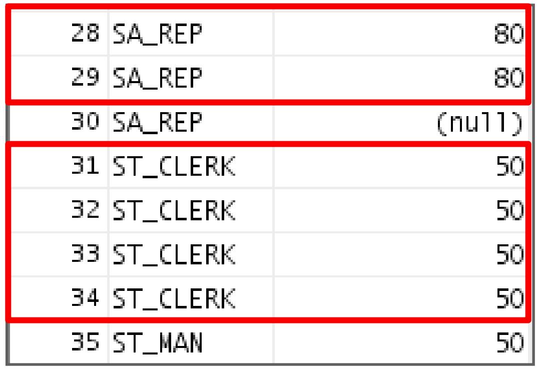
Display the jobs and departments of all current and previous employees.

```
SELECT job_id, department_id
FROM   employees
UNION ALL
SELECT job_id, department_id
FROM job_history
ORDER BY  job_id;
```



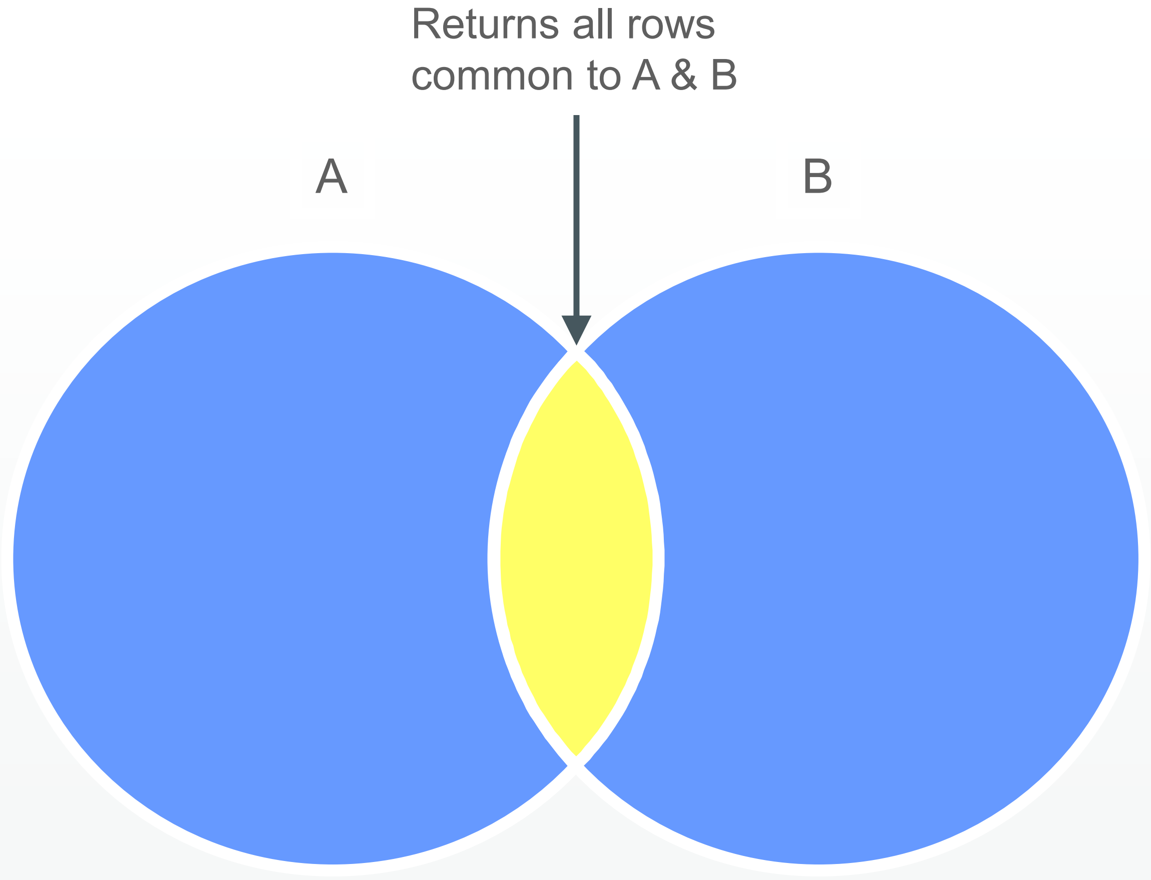
	JOB_ID	DEPARTMENT_ID
1	AC_ACCOUNT	110
2	AC_MGR	110
3	AD_ASST	10
4	AD_PRES	90
5	AD_PRES	90
6	AD_VP	90
7	AD_VP	80
8	AD_VP	90
9	AD_VP	90

...



28	SA_REP	80
29	SA_REP	80
30	SA_REP	(null)
31	ST_CLERK	50
32	ST_CLERK	50
33	ST_CLERK	50
34	ST_CLERK	50
35	ST_MAN	50

INTERSECT Operator

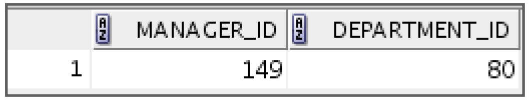


The INTERSECT operator returns rows that are common to both queries.

Using the INTERSECT Operator

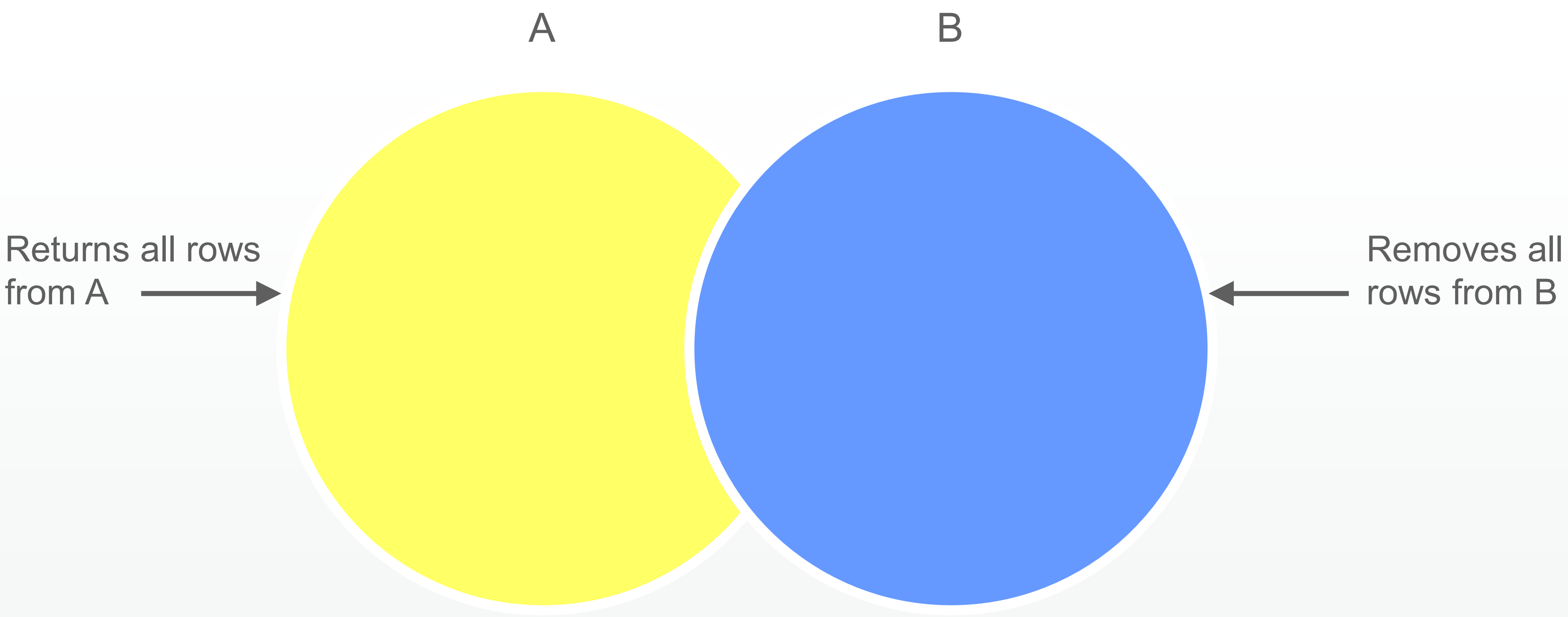
Display the common manager IDs and department IDs of current and previous employees.

```
SELECT  manager_id,department_id
FROM    employees
INTERSECT
SELECT manager_id,department_id
FROM job_history
```



	MANAGER_ID	DEPARTMENT_ID
1	149	80

EXCEPT Operator

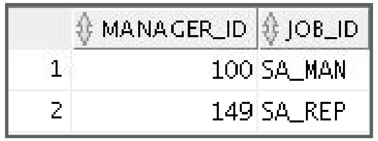


The MINUS operator returns all the distinct rows selected by the first query, but not present in the second query result set.

Using the EXCEPT Operator

Display the manager IDs and Job IDs of employees whose managers have never managed retired employees in the Sales department.

```
SELECT manager_id, job_id
FROM employees
WHERE department_id = 80
EXCEPT
SELECT manager_id, job_id
FROM job_history
WHERE department_id = 80;
```



	MANAGER_ID	JOB_ID
1	100	SA_MAN
2	149	SA_REP

Matching SELECT Statements

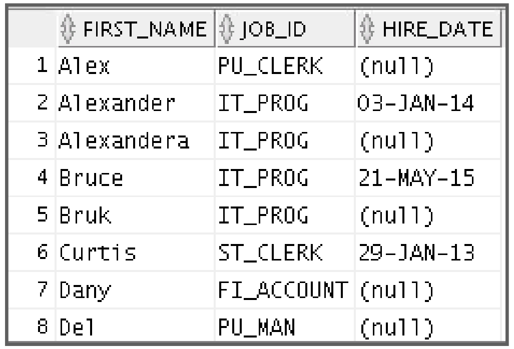
You must match the data type (using the TO_CHAR function or any other conversion functions) when columns do not exist in one or the other table.

```
SELECT location_id, department_name "Department",
   NULL "Warehouse location"
FROM departments
UNION
SELECT location_id, NULL "Department",
   state_province
FROM locations;
```

Matching the SELECT Statement: Example

Using the UNION operator, display the employee name, job ID, and hire date of all employees.

```
SELECT  FIRST_NAME, JOB_ID, hire_date "HIRE_DATE"
FROM employees
UNION
SELECT FIRST_NAME, JOB_ID, NULL "HIRE_DATE"
FROM job_history;
```



	FIRST_NAME	JOB_ID	HIRE_DATE
1	Alex	PU_CLERK	(null)
2	Alexander	IT_PROG	03-JAN-14
3	Alexandera	IT_PROG	(null)
4	Bruce	IT_PROG	21-MAY-15
5	Bruk	IT_PROG	(null)
6	Curtis	ST_CLERK	29-JAN-13
7	Dany	FI_ACCOUNT	(null)
8	Del	PU_MAN	(null)

...