Basic SQL

Week 4

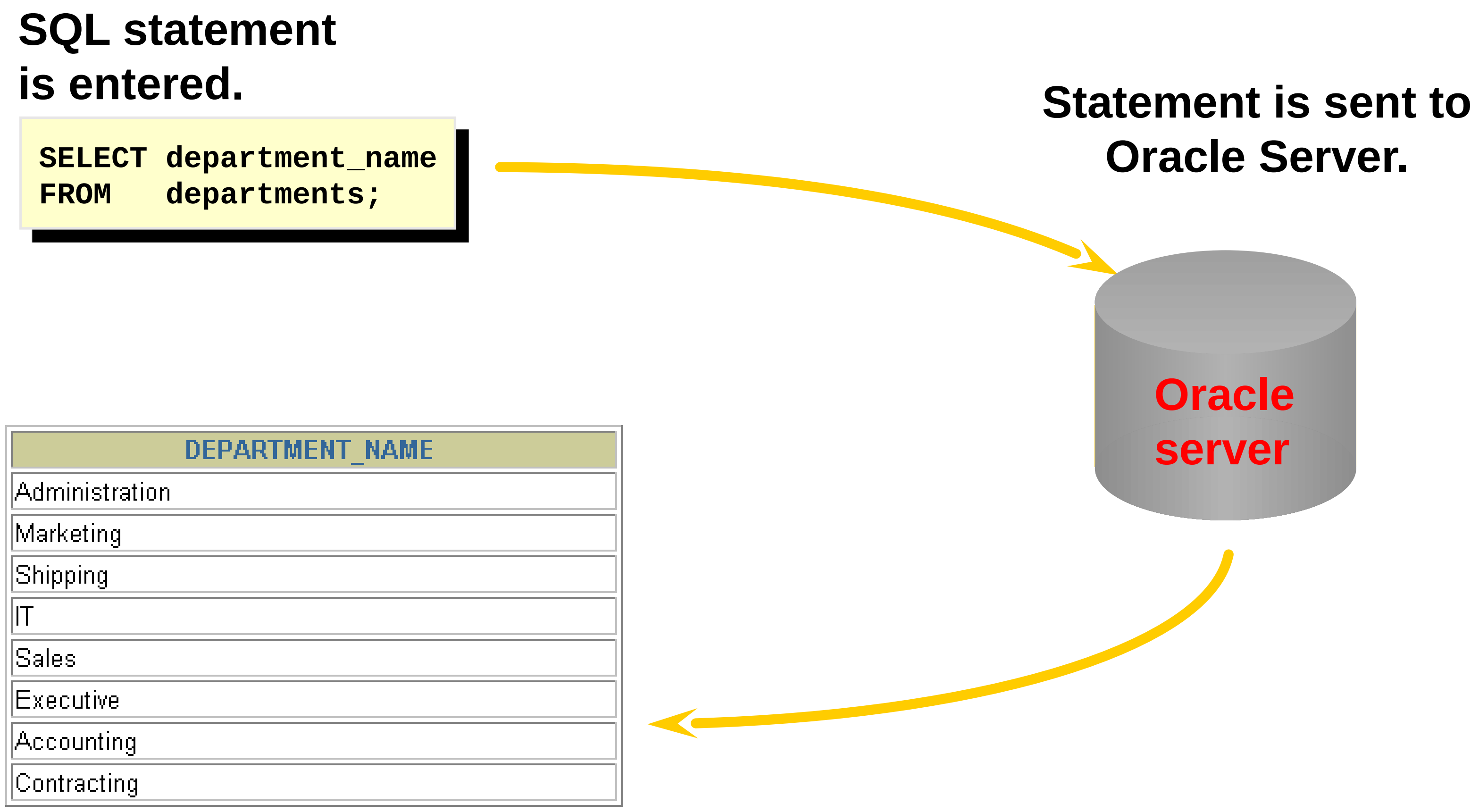
Structured Query Language (SQL)

- SQL is structured Query Language which is a computer language for storing, manipulating and retrieving data stored in relational database.
- SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server uses SQL as standard database language.

Why do we need SQL?

- Allow users to access data in relational database management systems.
- Allow users to describe the data.
- Allow users to define the data in database and manipulate that data.
- Allow users to create and drop databases and tables.
- Allow users to create view, stored procedure, functions in a database.
- Allow users to set permissions on tables, procedures, and views

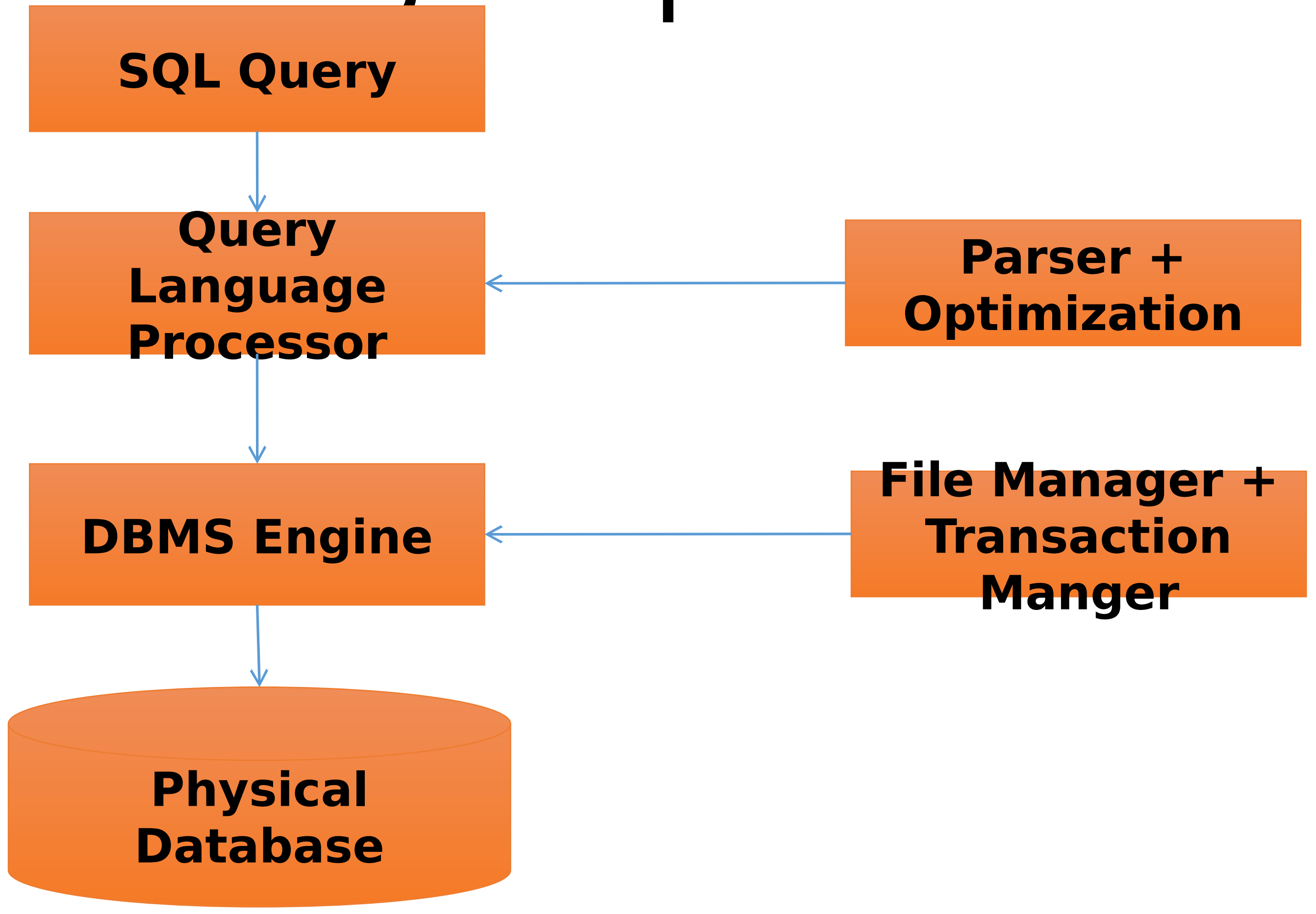
Communicating with the RDBMS


SQL statement is entered.
Statement is sent to Oracle Server.
Oracle server

```
SELECT department_name
FROM   departments;
```

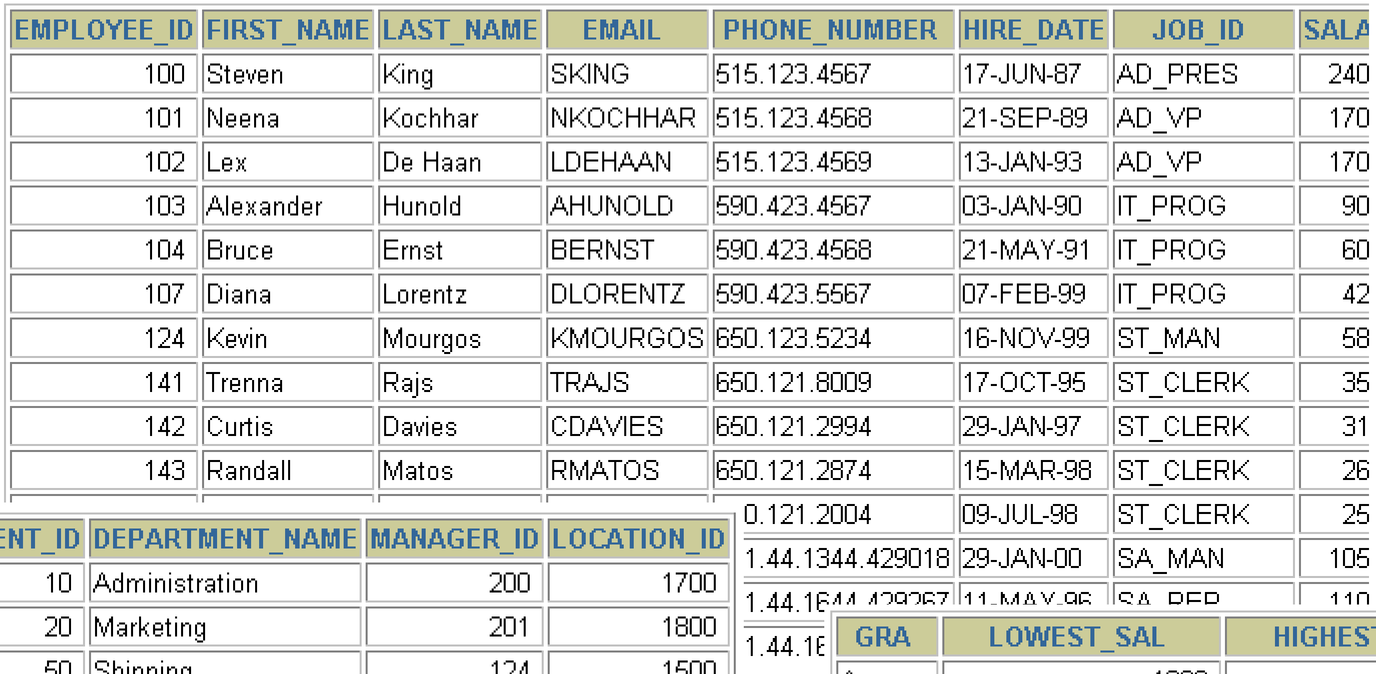
DEPARTMENT_NAME
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Contracting

SQL Processing Steps

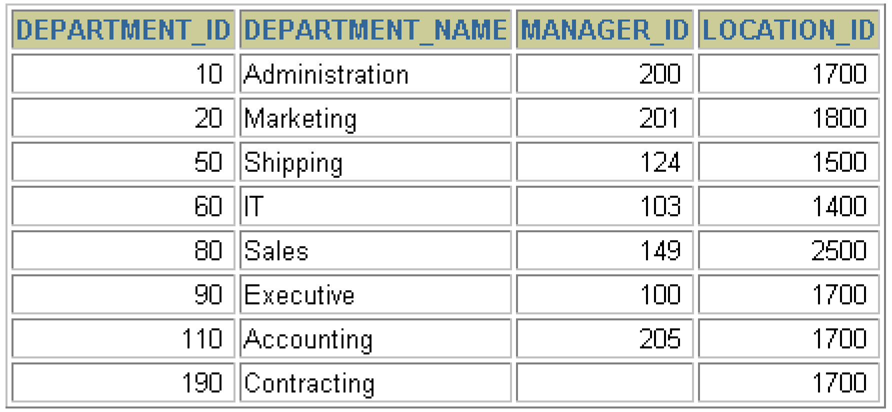


Tables used in this course

EMPLOYEES

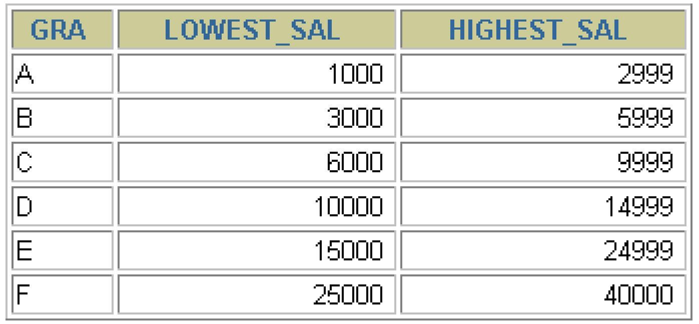


EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
				0.121.2004	09-JUL-98	ST_CLERK	25
				1.44.1344.429018	29-JAN-00	SA_MAN	105
				1.44.1644.429267	11-MAY-96	SA_REP	110
				1.44.16			



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

DEPARTMENTS



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

JOB_GRADES

SQL Statements

Data manipulation language (DML)

Data definition language (DDL)

Transaction control language (TCL)

Data control language (DCL)

The CREATE TABLE Statement

- You must have:
  - CREATE TABLE privilege

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

- You specify:
  - Table name
  - Column name, data type, size and constraint if any

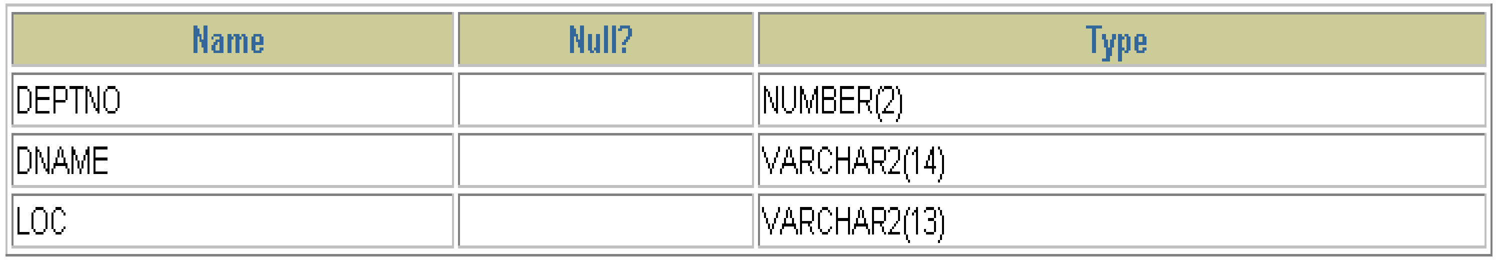
Creating Tables

Dept(deptno:number, dname:text(14), loc:text(13))

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

- Confirm table creation.

```
DESCRIBE dept
```



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

Dropping a Table

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- You *cannot* roll back the `DROP TABLE` statement.

```
DROP TABLE dept;
Table dropped.
```

The DEFAULT Option

- Specify a default value for a column during

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudo column are illegal values.
- The default data type must match the column data type.

Setting DEFAULT Values

```
CREATE TABLE product(
       prod_no    NUMBER(2),
       Prod_name  VARCHAR2(14),
       Mfg_date   DATE DEFAULT SYSDATE,
       Prod_price NUMBER(4,2) DEFAULT 0.0
);
Table created.
```

What Are Constraints?

- Constraints enforce rules at the table & column level.
- Constraints prevent the deletion of a table/data if there are dependencies.
- One should be familiar with the following:
  - `When to create a constraint?`
  - `Level of creation`
  - `Types of constraint`
  - `How to name a constraint?`

Constraint Types

- NOT NULL [used when field is mandatory]
- UNIQUE [used when no duplicates are required]
- CHECK [used when some condition is needed to apply]
- PRIMARY KEY [each relation have one primary key]
- FOREIGN KEY [used to get data from other table]

When to create a constraint?

At the time of table creation using CREATE TABLE command

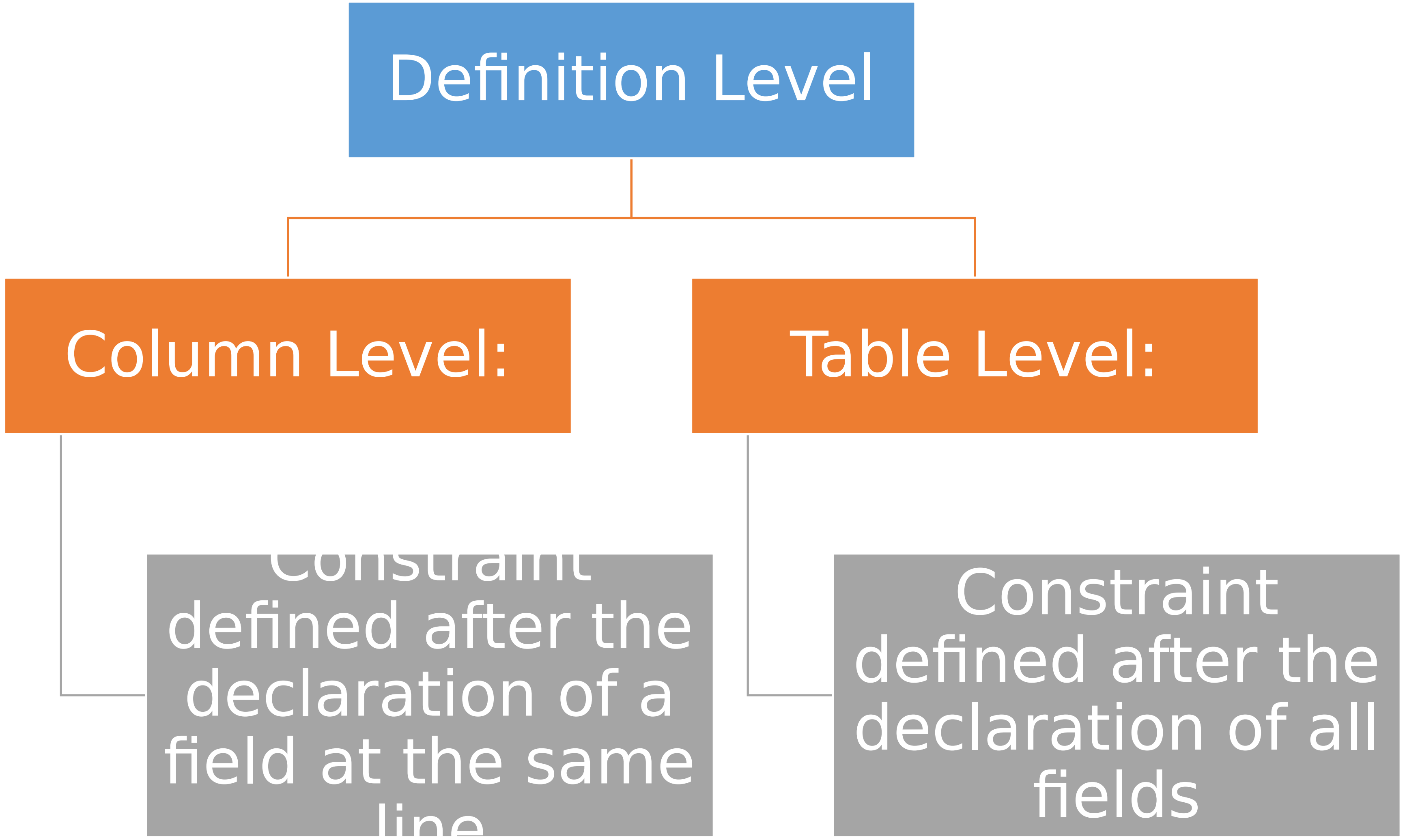
After table creation using ALTER TABLE command

[will cover in next class]

CREATE TABLE command

```
CREATE TABLE employees(
    employee_id    NUMBER(6),
    last_name      VARCHAR2(25),
    salary         NUMBER(8,2),
    commission_pct NUMBER(2,2),
    hire_date      DATE
)
```

Where to define a Constraint?


Definition Level
Column Level:
Table Level:
Constraint defined after the declaration of a field at the same line
Constraint defined after the declaration of all fields

Constraints: Basic Syntax

```
CREATE TABLE [schema.]table
            (column datatype [column_constraint],
             ...
             [table_constraint]);
```

- Column level constraint

```
column  datatype [CONSTRAINT C-name] constraint_type,
```

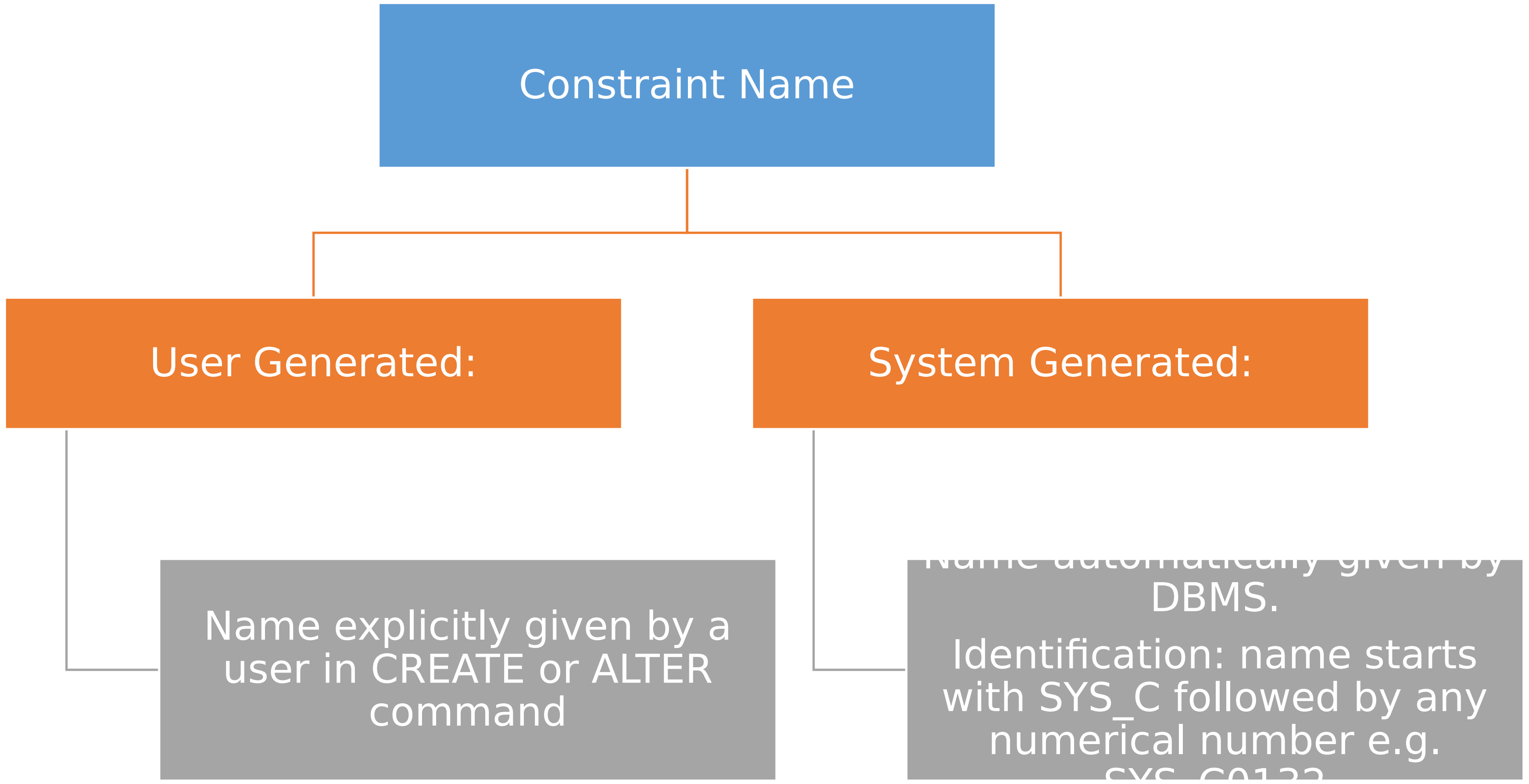
- Table level constraint

```
column,...
[CONSTRAINT constraint_name] constraint_type (column)
```

CREATE TABLE command

```
CREATE TABLE employees(
    employee_id    NUMBER(6),
    last_name      VARCHAR2(25) NOT NULL,
    salary         NUMBER(8,2),
    commission_pct NUMBER(2,2),
    hire_date      DATE,
  or NOT NULL (last_name)
)
```

Naming a Constraint



Constraint Name

User Generated:

Name explicitly given by a user in CREATE or ALTER command

System Generated:

Name automatically given by DBMS.

Identification: name starts with SYS_C followed by any numerical number e.g. SYS_C0132

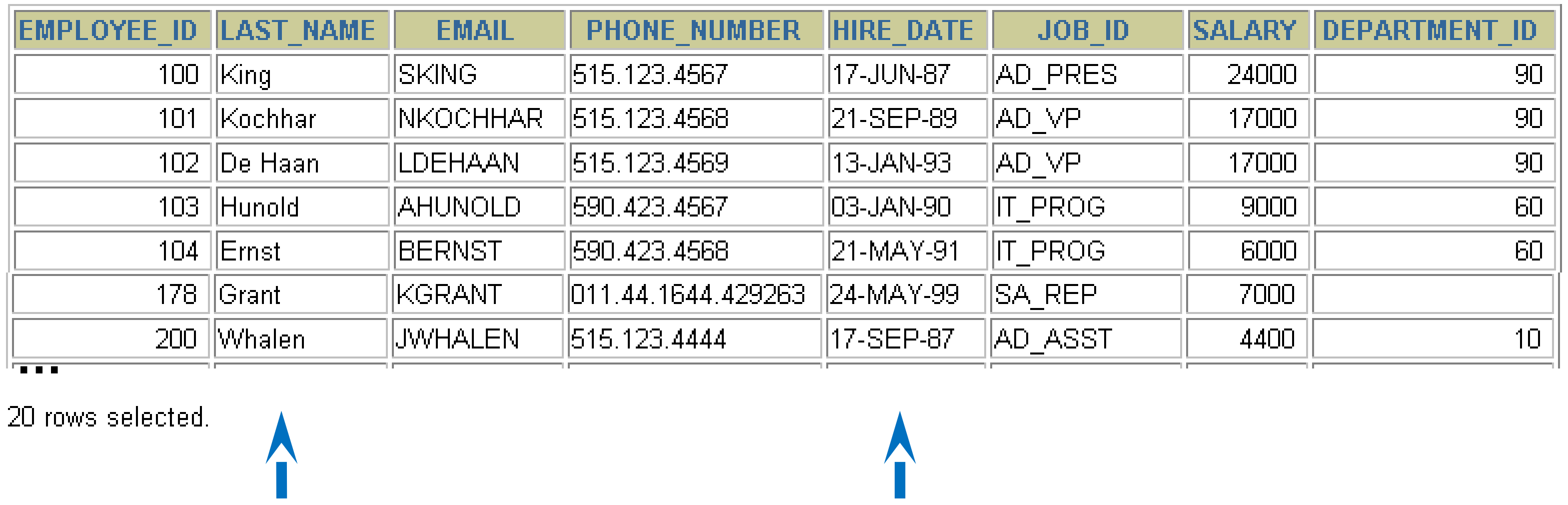
Why a constraint name?

- The name of a constraint is required when
  - You need to drop a constraint from a column or table
  - You need to modify the existing constraint applied on a specific column or table

- Where to find the details about constraints?
  - `USER_CONSTRAINTS` table keeps track of all constraints defined by user in any table
  - `USER_CONS_Columns` table keeps track of all constraints defined by user at any column in any table

The NOT NULL Constraint

Ensures that null values are not permitted for the column:



EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

20 rows selected.

NOT NULL constraint (No row can contain a null value for this column.)

NOT NULL constraint

The NOT NULL Constraint

Is defined at the column level:

```
CREATE TABLE employees(
    employee_id    NUMBER(6),
    last_name      VARCHAR2(25) NOT NULL,
    salary         NUMBER(8,2),
    commission_pct NUMBER(2,2),
    hire_date      DATE  CONSTRAINT emp_hiredate_nn NOT NULL
)
```

System named

User named

Viewing Constraints

Show all constraints defined by user on employees table

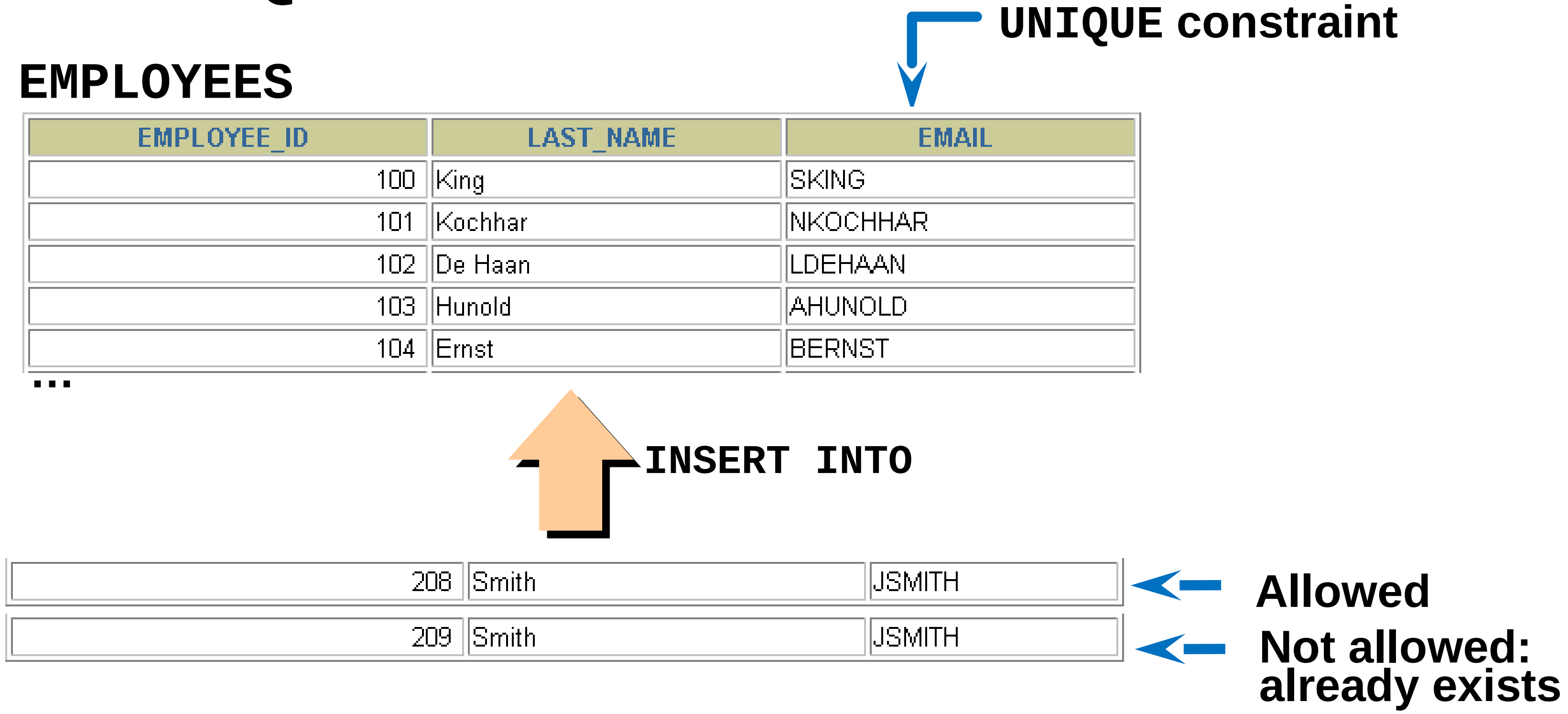
```
SELECT   constraint_name, constraint_type,
         search_condition
FROM     user_constraints
WHERE    table_name = 'EMPLOYEES';
```

Viewing the Columns Associated with Constraints

Find the columns of employee table where constraints are applied.

```
SELECT   constraint_name, column_name
FROM     user_cons_columns
WHERE    table_name = 'EMPLOYEES';
```

The UNIQUE Constraint



EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL
100	King	SKING
101	Kochhar	NKOCHHAR
102	De Haan	LDEHAAN
103	Hunold	AHUNOLD
104	Ernst	BERNST

...

208	Smith	JSMITH
209	Smith	JSMITH

The UNIQUE Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

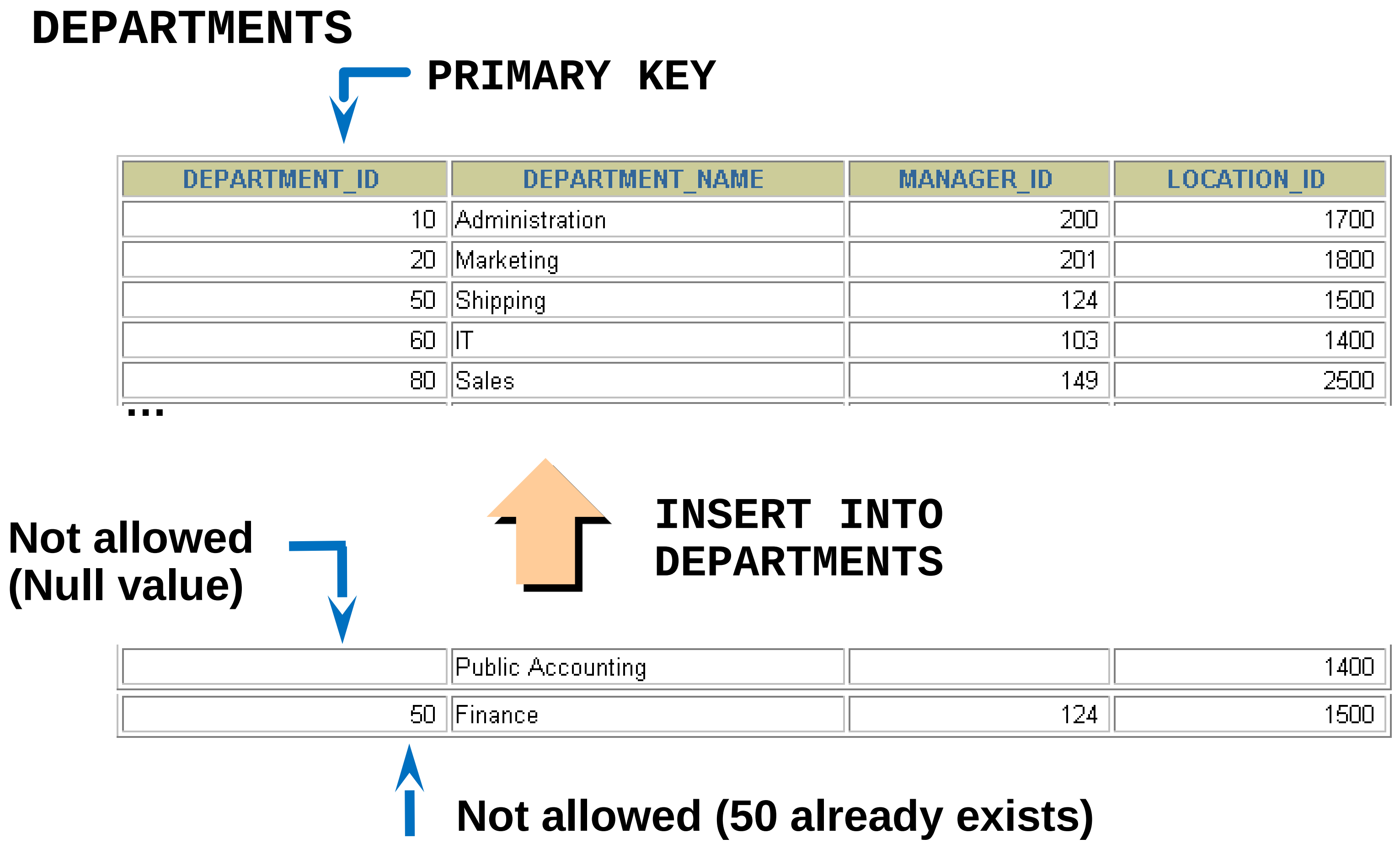
```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25) UNIQUE,
...
);
```

The CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

```
..., salary  NUMBER(2)
     CONSTRAINT emp_salary_min
            CHECK (salary > 0),...
```

The PRIMARY KEY Constraint



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

...

	Public Accounting		1400
50	Finance	124	1500

The PRIMARY KEY Constraint

Defined at table level

```
CREATE TABLE   departments(
    department_id        NUMBER(4),
    department_name      VARCHAR2(30)
      CONSTRAINT dept_name_nn NOT NULL,
    manager_id           NUMBER(6),
    location_id          NUMBER(4),
    CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

Defined at Column level

```
CREATE TABLE   departments(
    department_id        NUMBER(4) PRIMARY KEY,
    department_name      VARCHAR2(30)
      CONSTRAINT dept_name_nn NOT NULL,
    manager_id           NUMBER(6),
    location_id          NUMBER(4)
);
```

The Composite PRIMARY KEY Constraint

Defined at table level

```
CREATE TABLE   CourseRegistration(
    Reg_No         NUMBER(4),
    Course_Code    VARCHAR2(6),
    Semester       Number(1),
    PRIMARY KEY(Reg_No, Course_Code)
);
```

Composite PRIMARY KEY

Defined at Column level

```
CREATE TABLE   CourseRegistration(

    Reg_No         NUMBER(4) PRIMARY KEY,

    Course_Code    VARCHAR2(6) PRIMARY KEY,

    Semester       Number(1)

);
```

PRIMARY KEY

PRIMARY KEY

This table has two simple primary keys

The FOREIGN KEY Constraint

DEPARTMENTS

PRIMARY KEY

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

…

EMPLOYEES

FOREIGN KEY

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

…

INSERT INTO

200	Ford	9	Not allowed (9 does not exist)
201	Ford	60	Allowed

The FOREIGN KEY Constraint

Defined at column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    department_id    NUMBER(4)
    REFERENCES departments);
```

Defined at table level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
      REFERENCES departments(department_id));
```

`FOREIGN KEY` Constraint Keywords

- `FOREIGN KEY`: Defines the column in the child table at the table constraint level
- `REFERENCES`: Identifies the table and column in the parent table
- `ON DELETE CASCADE`: Deletes the dependent rows in the child table when a row in the parent table is deleted.
- `ON DELETE SET NULL`: Converts dependent foreign key values to null

The ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns.

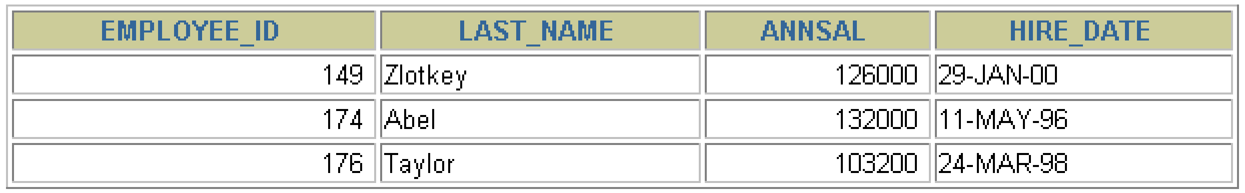
```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP       (column);
```

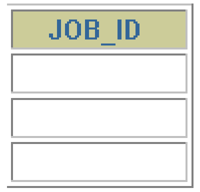
Adding a Column

DEPT



EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
149	Zlotkey	126000	29-JAN-00
174	Abel	132000	11-MAY-96
176	Taylor	103200	24-MAR-98

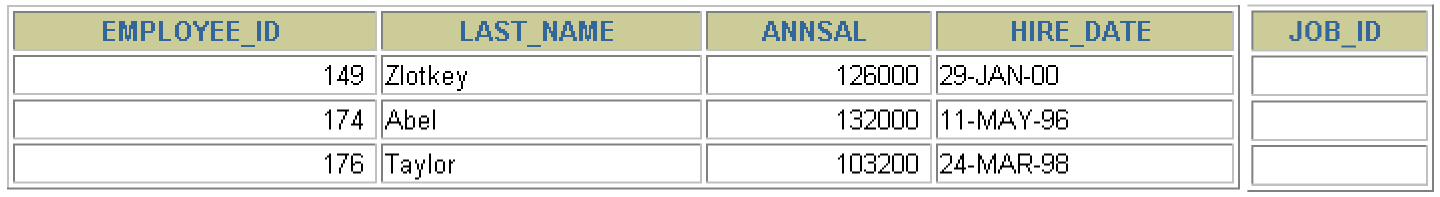
New column



JOB_ID

“Add a new column to DEPT table.”

DEPT



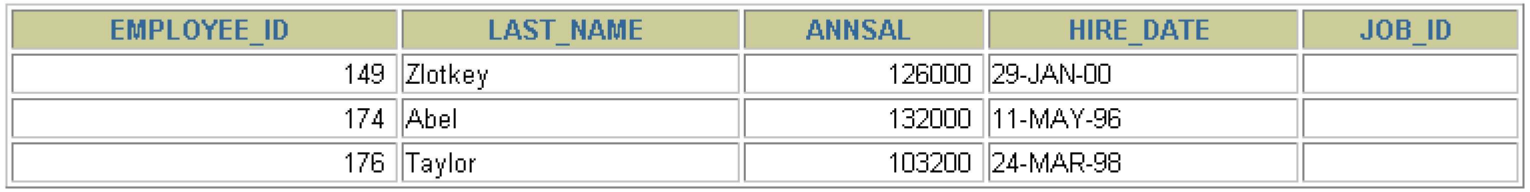
EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
149	Zlotkey	126000	29-JAN-00	
174	Abel	132000	11-MAY-96	
176	Taylor	103200	24-MAR-98	

Adding a Column

- You use the ADD clause to add columns.

```
ALTER TABLE dept
ADD         (job_id VARCHAR2(9));
Table altered.
```

- The new column becomes the last column.



EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
149	Zlotkey	126000	29-JAN-00	
174	Abel	132000	11-MAY-96	
176	Taylor	103200	24-MAR-98	

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE   dept
MODIFY        (last_name VARCHAR2(30));
Table altered.
```

- A change to the default value affects only subsequent insertions to the table.

Dropping a Column

Use the `DROP COLUMN` clause to drop columns you no longer need from the table.

```
ALTER TABLE  dept
DROP COLUMN  job_id;
Table altered.
```

Changing the Name of an Object

- To change the name of a table, view, sequence, or synonym, you execute the RENAME statement.

```
RENAME dept TO department;
Table renamed.
```

- You must be the owner of the object.

Truncating a Table

- The TRUNCATE TABLE statement:
  - Removes all rows from a table
  - Releases the storage space used by that table

```
TRUNCATE TABLE department;
Table truncated.
```

- You cannot roll back row removal when using TRUNCATE.
- Alternatively, you can remove rows by using the DELETE statement.

DELETE FROM table-name;

Adding a Constraint Syntax

Use the ALTER TABLE statement to:

- Add or drop a constraint, but not modify its structure
- Enable or disable constraints
- Add a NOT NULL constraint by using the MODIFY clause

```
ALTER TABLE  table
ADD [CONSTRAINT constraintName] type (column);
```

Adding a Constraint

Add a FOREIGN KEY constraint to the EMPLOYEES table indicating that a manager must already exist as a valid employee in the EMPLOYEES table.

```
ALTER TABLE     employees
ADD CONSTRAINT  emp_manager_fk
  FOREIGN KEY(manager_id)
  REFERENCES employees(employee_id);
Table altered.
```

Dropping a Constraint

- Remove the manager constraint from the EMPLOYEES table.

```
ALTER TABLE      employees
DROP CONSTRAINT  emp_manager_fk;
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPARTMENTS table and drop the associated FOREIGN KEY constraint on the EMPLOYEES.DEPARTMENT_ID column.

```
ALTER TABLE   departments
DROP PRIMARY KEY CASCADE;
Table altered.
```

Disabling Constraints

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable dependent integrity constraints.

```
ALTER TABLE          employees
DISABLE CONSTRAINT   emp_emp_id_pk CASCADE;
Table altered.
```

Enabling Constraints

- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

```
ALTER TABLE          employees
ENABLE CONSTRAINT    emp_emp_id_pk;
Table altered.
```

- A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE key or PRIMARY KEY constraint.

Cascading Constraints

- The `CASCADE CONSTRAINTS` clause is used along with the `DROP COLUMN` clause.
- The `CASCADE CONSTRAINTS` clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The `CASCADE CONSTRAINTS` clause also drops all multicolumn constraints defined on the dropped columns.

Cascading Constraints

Example:

```
ALTER TABLE test1
DROP (pk) CASCADE CONSTRAINTS;
Table altered.
```

```
ALTER TABLE test1
DROP (pk, fk, col1) CASCADE CONSTRAINTS;
Table altered.
```

Guidelines for writing SQL

- SQL statements are not case sensitive.
- SQL statements can be 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.

Basic SELECT statement

- Syntax

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

- SELECT identifies which columns
- FROM identifies which table

Selecting all columns

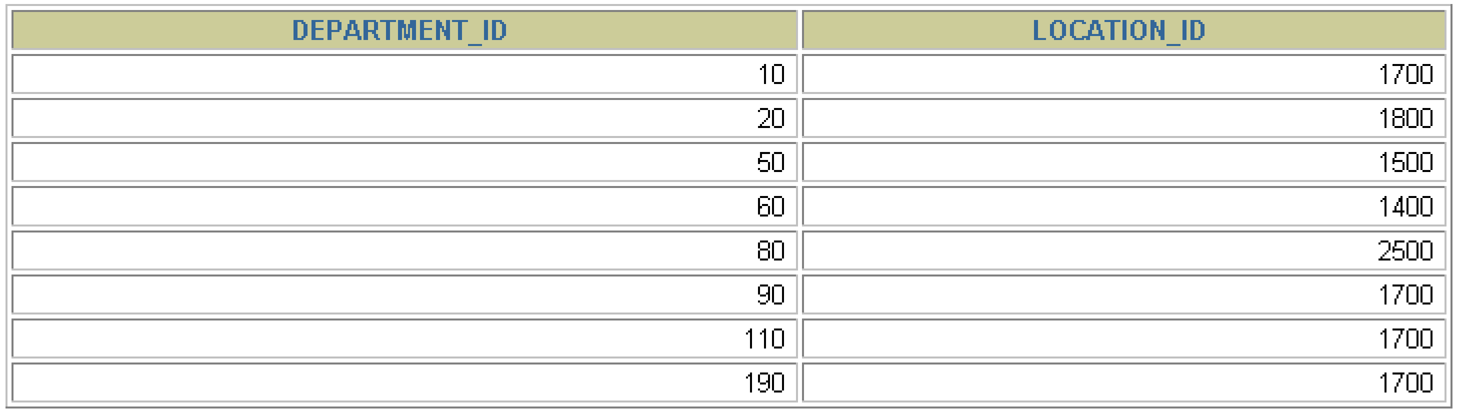
```
SELECT *
FROM   departments;
```

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

8 rows selected.

Selecting specific columns

```
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

8 rows selected.

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 alias
  - Requires double quotation marks if it contains spaces or special characters or is case sensitive

Using column aliases

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

NAME	COMM
King	
Kochhar	
De Haan	

...

20 rows selected.

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

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

...

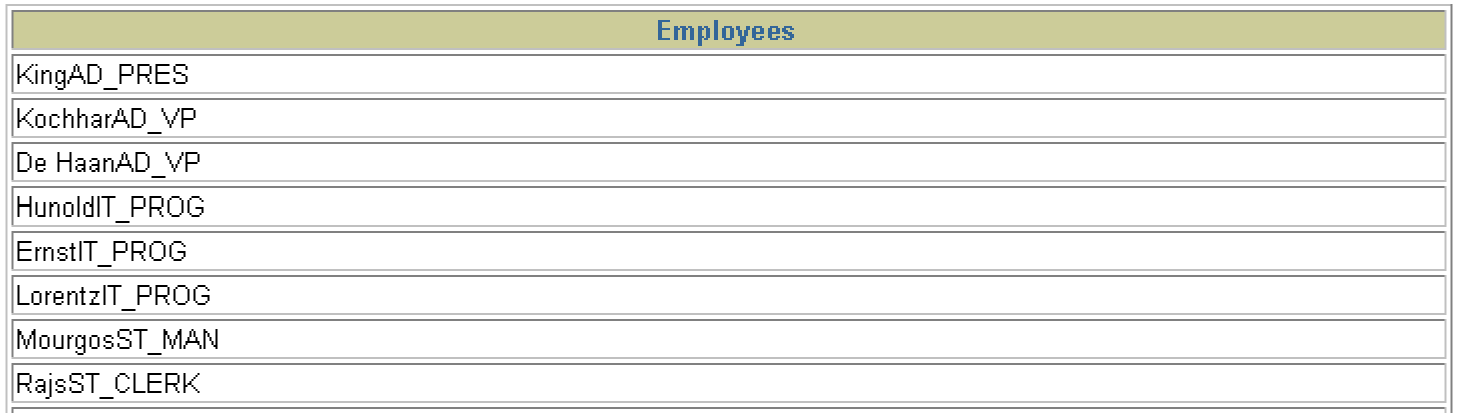
20 rows selected.

Concatenation operator

- A concatenation operator:
  - Concatenates columns or character strings to other columns
  - Is represented by two vertical bars (||)
  - Creates a resultant column that is a character expression

Using concatenation operator

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



Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
ErnstIT_PROG
LorentzIT_PROG
MourgosST_MAN
RajsST_CLERK

...

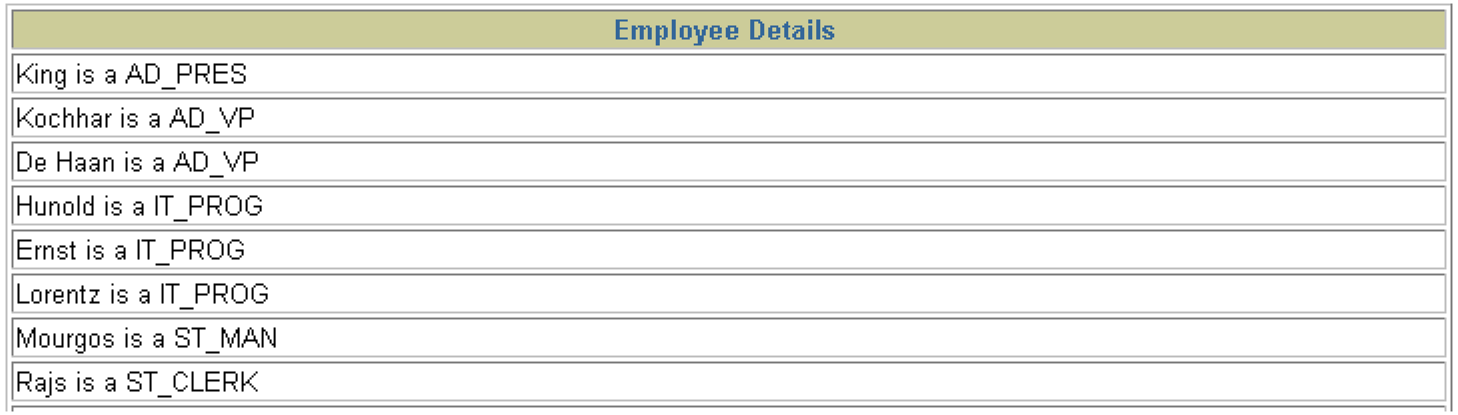
20 rows selected.

Literal character strings

- A literal is a character, a number, or a date included in the SELECT list.
- 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
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG
Mourgos is a ST_MAN
Rajs is a ST_CLERK

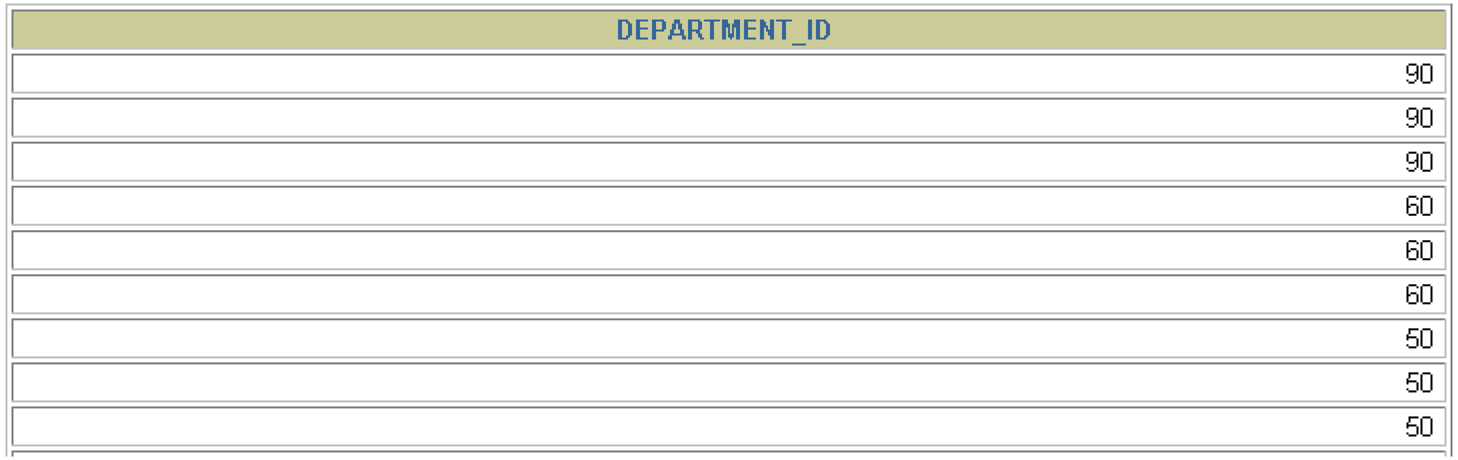
...

20 rows selected.

Duplicate rows

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

```
SELECT department_id
FROM   employees;
```



DEPARTMENT_ID
90
90
90
60
60
60
50
50
50

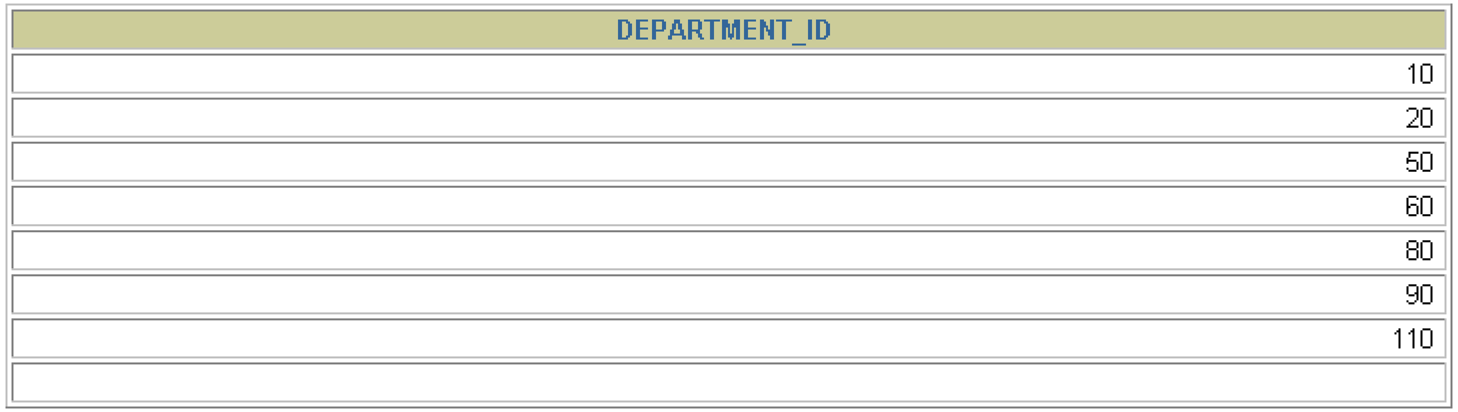
…

20 rows selected.

Eliminating duplicate rows

- Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

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



DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

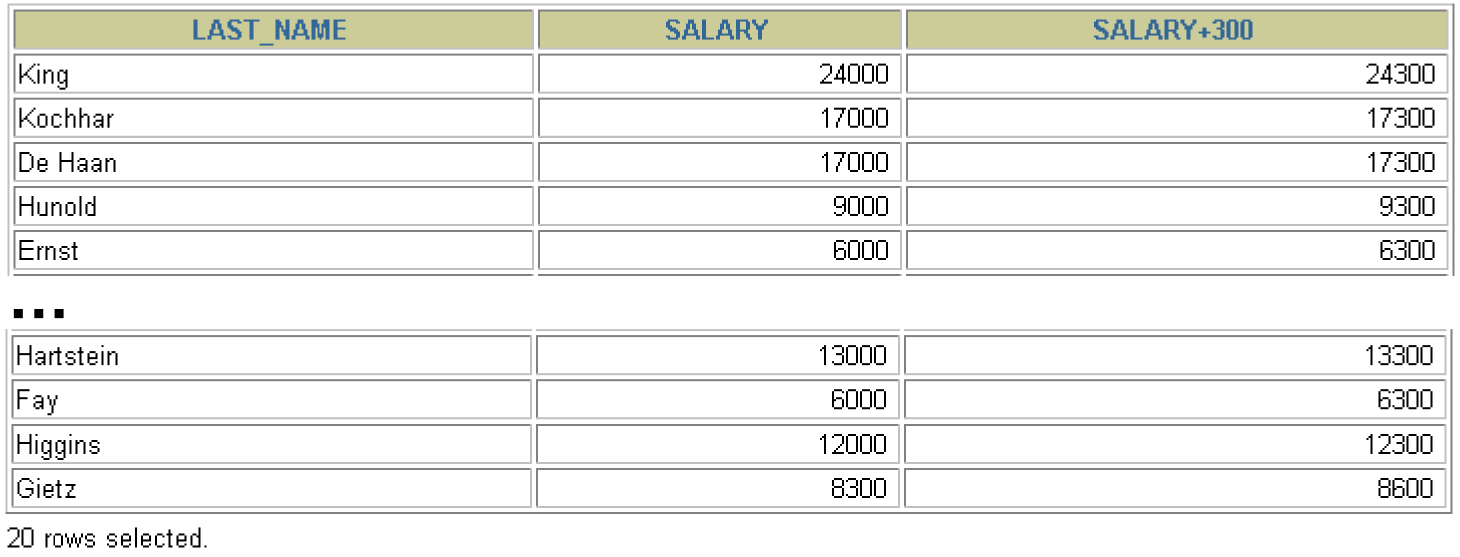
Arithmetic Expressions

- 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
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300

...

Hartstein	13000	13300
Fay	6000	6300
Higgins	12000	12300
Gietz	8300	8600

20 rows selected.

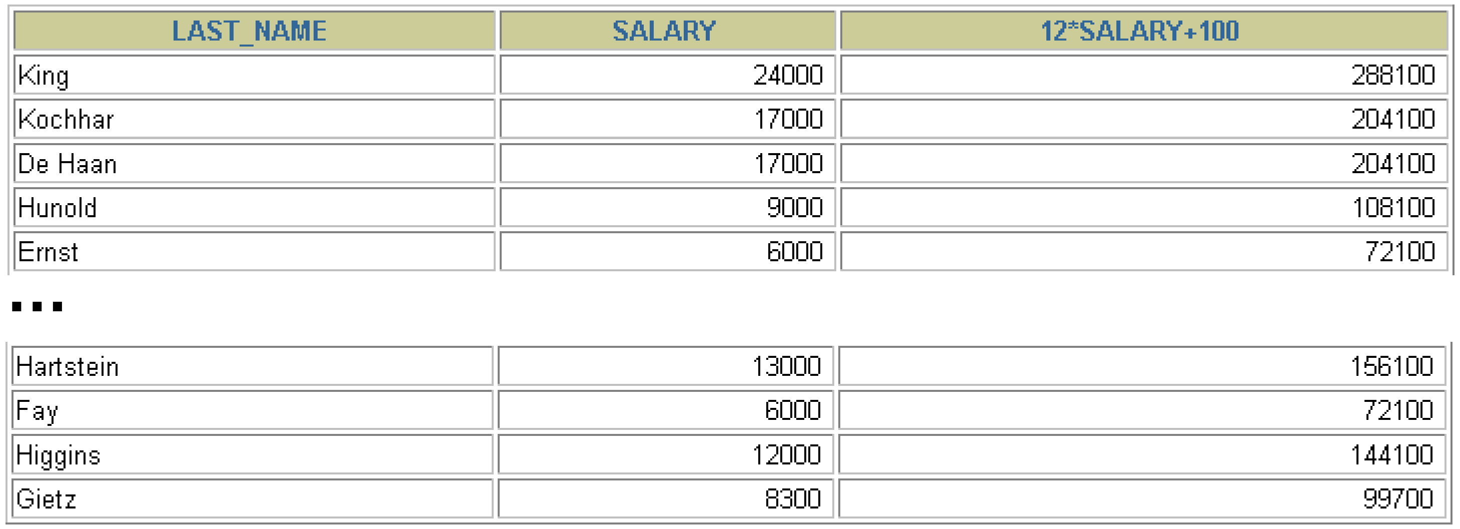
Operators precedence

***   /   +   -**

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

Operator precedence

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



LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernst	6000	72100

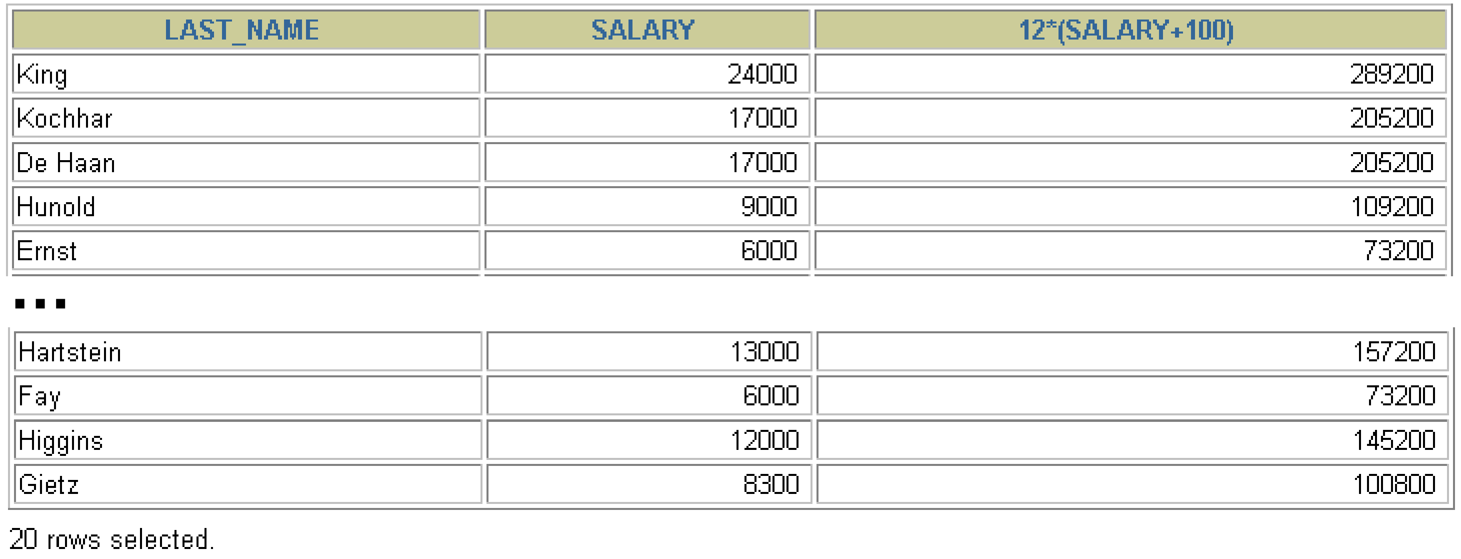
...

Hartstein	13000	156100
Fay	6000	72100
Higgins	12000	144100
Gietz	8300	99700

20 rows selected.

Using parenthesis

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



LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200

...

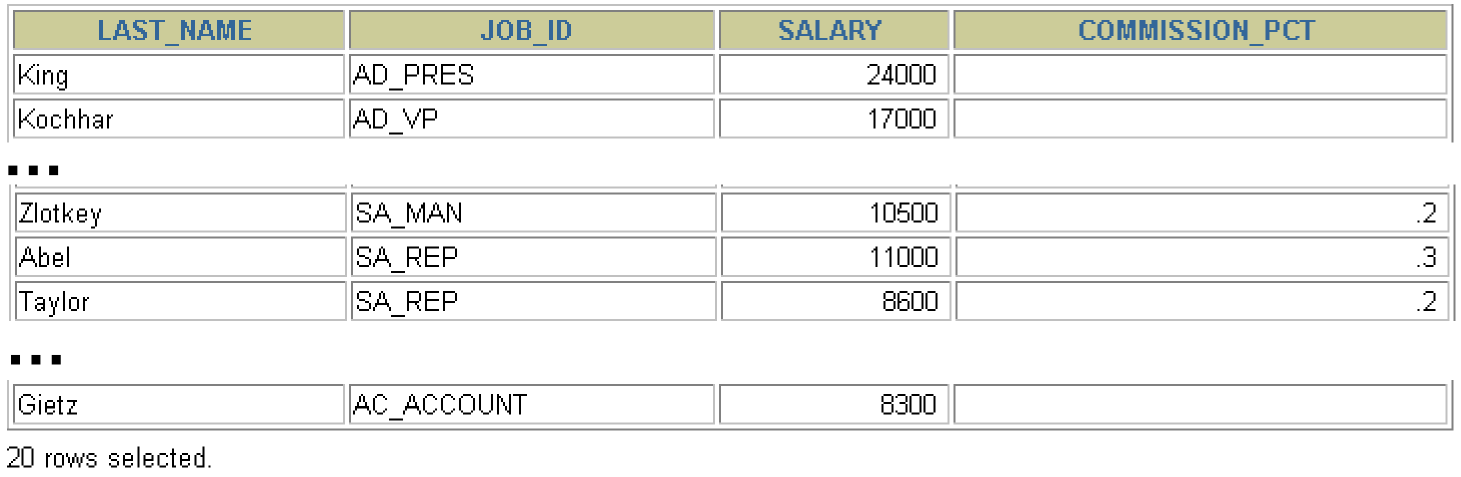
Hartstein	13000	157200
Fay	6000	73200
Higgins	12000	145200
Gietz	8300	100800

20 rows selected.

Defining a NULL value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A 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
King	AD_PRES	24000	
Kochhar	AD_VP	17000	

...

Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2

...

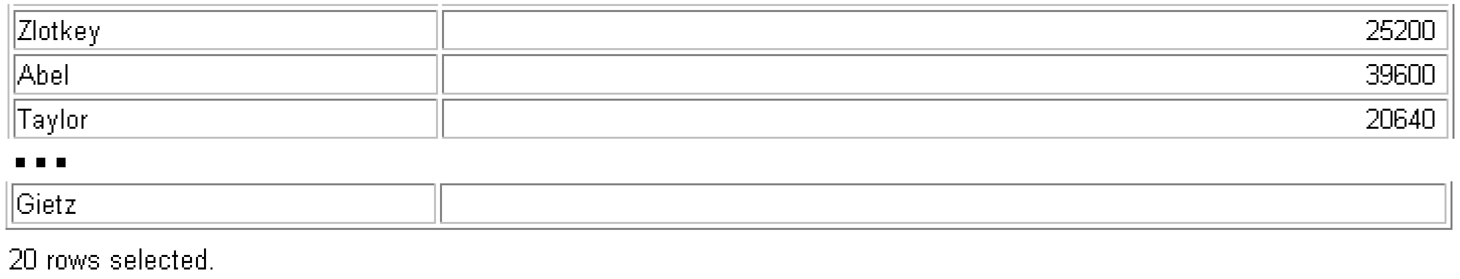
Gietz	AC_ACCOUNT	8300	

20 rows selected.

NULL values in Arithmetic Expressions

- Arithmetic expressions containing a null value evaluate to null.

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



Zlotkey	25200
Abel	39600
Taylor	20640

...

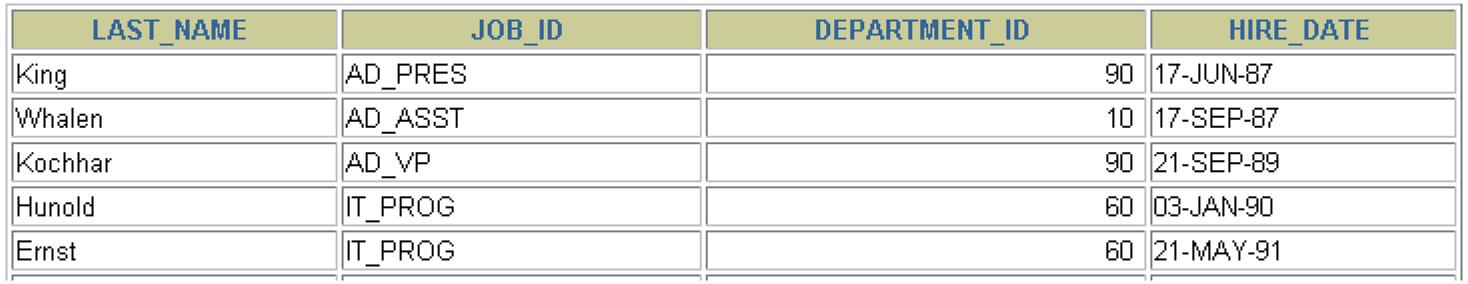
Gietz	

20 rows selected.

Sorting Rows

- Sort 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 ;
```



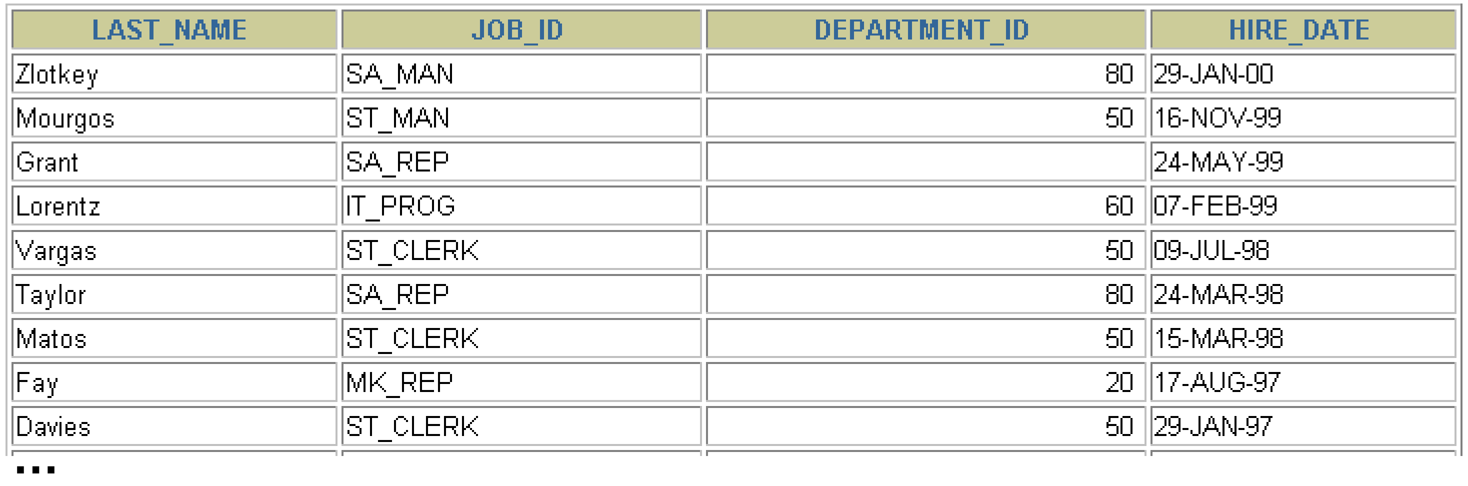
LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

…

20 rows selected.

Sorting in Descending Order

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



LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97

...

20 rows selected.

Sorting by Column Alias

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

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000

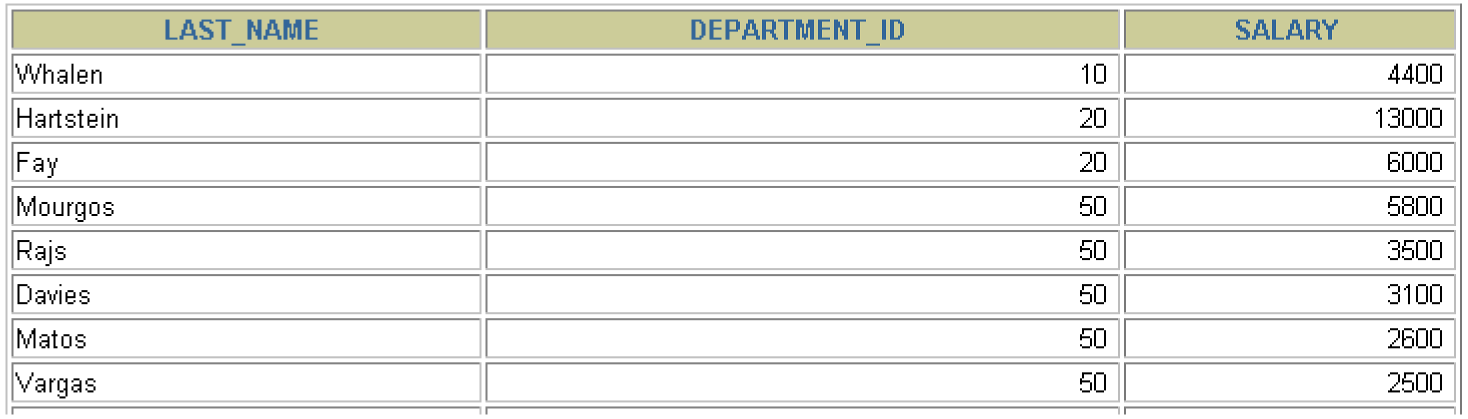
...

20 rows selected.

Sorting by Multiple Columns

- The order of ORDER BY list is the order of sort.

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



LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Davies	50	3100
Matos	50	2600
Vargas	50	2500

...

20 rows selected.

- You can sort by a column that is not in the SELECT list.

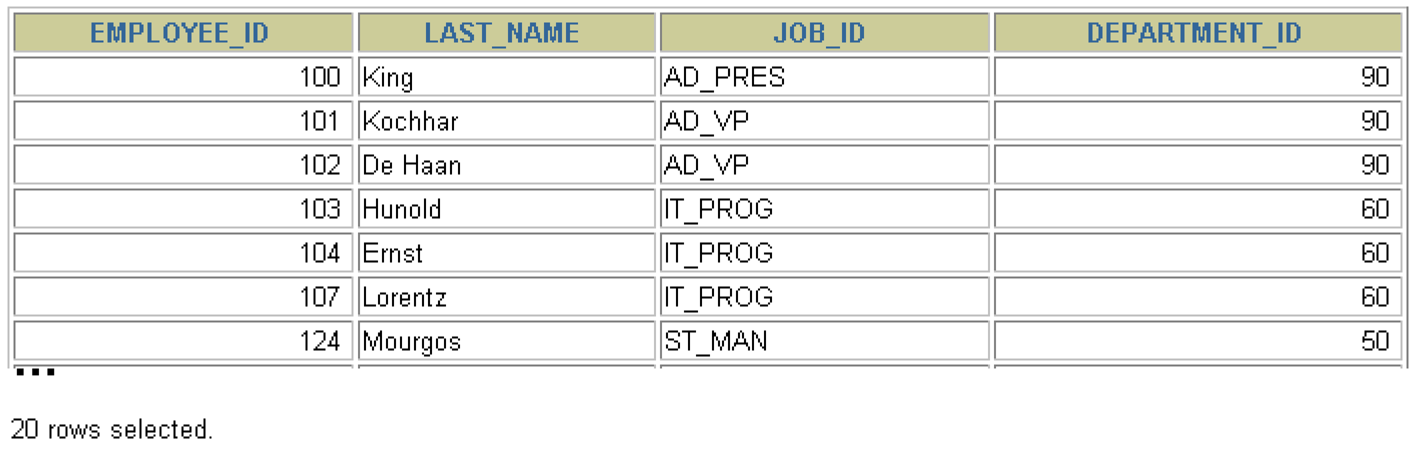
Outline

In this lecture, you will learn:

- SQL SELECT
  - Row Filtering
- Operators
  - Comparison operators
  - String & Set operators
  - Logical operators

Limiting Rows Using a Selection

EMPLOYEES

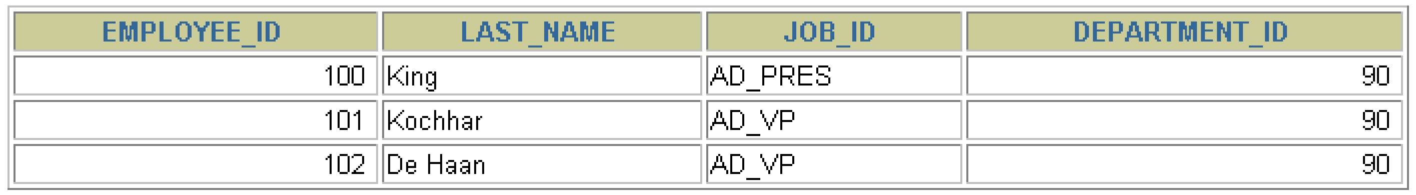


EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60
124	Mourgos	ST_MAN	50

…

20 rows selected.

“retrieve all employees in department 90”



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

Limiting the Rows Selected

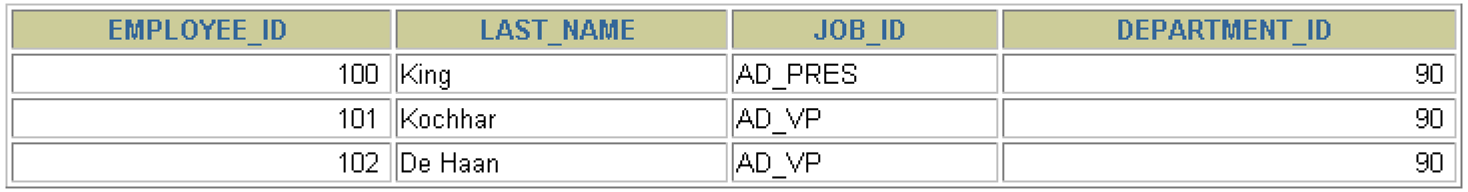
- Restrict the rows returned by using the WHERE clause.

```
SELECT   *|{[DISTINCT] column|expression [alias],...}
FROM     table
[WHERE   condition(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
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks.
- Character values are case sensitive, and date values are format sensitive.
- The default date format is DD-MON-RR.

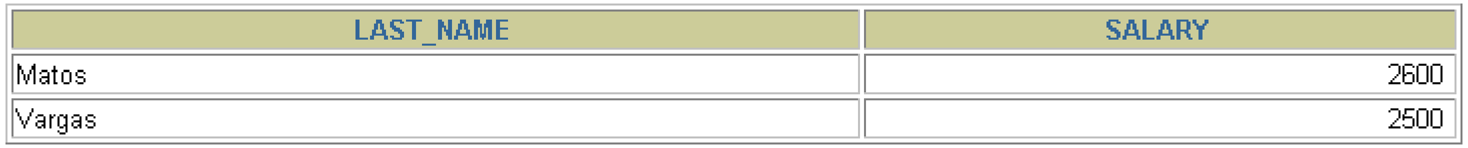
```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen';
```

Comparison Conditions

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

Using Comparison Conditions

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



LAST_NAME	SALARY
Matos	2600
Vargas	2500

Other Comparison Conditions

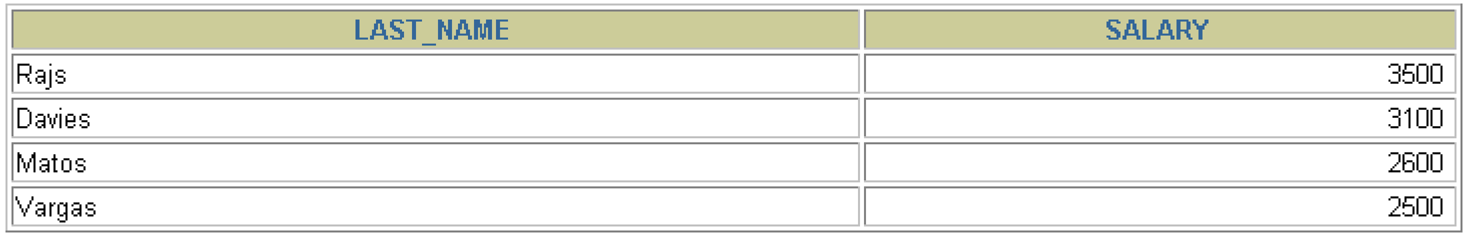
Operator	Meaning
`BETWEEN ...AND...`	Between two values (inclusive),
`IN(set)`	Match any of a list of values
`LIKE`	Match a character pattern
`IS NULL`	Is a null value

Using the BETWEEN Condition

Use the BETWEEN condition 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

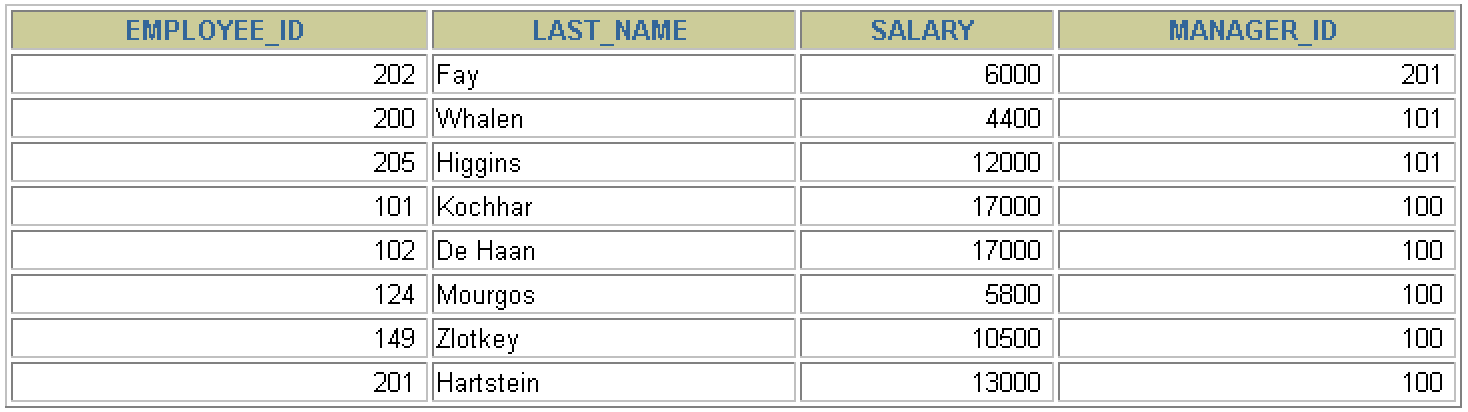


LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Using the IN Condition

Use the IN membership condition 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
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

Using the LIKE Condition

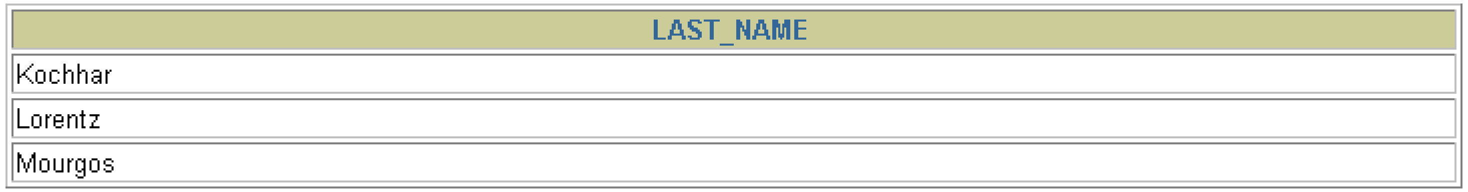
- Use the LIKE condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - % denotes zero or many characters.
  - _ denotes one character.

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

Using the LIKE Condition

- You can combine pattern-matching characters.

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



LAST_NAME
Kochhar
Lorentz
Mourgos

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

Using the NULL Conditions

Test for nulls with the `IS NULL` operator.

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

LAST_NAME	MANAGER_ID
King	

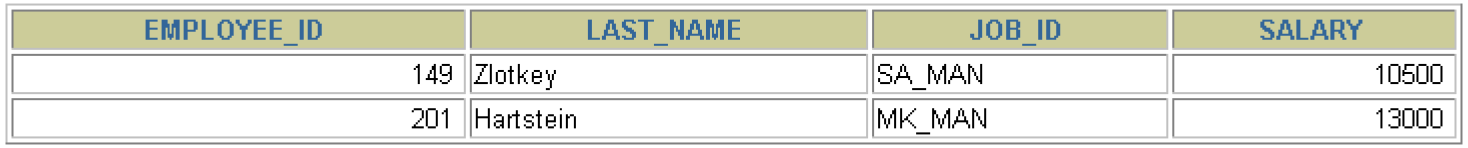
Logical Conditions

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 following condition is false

Using the AND Operator

AND requires both conditions to be true.

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

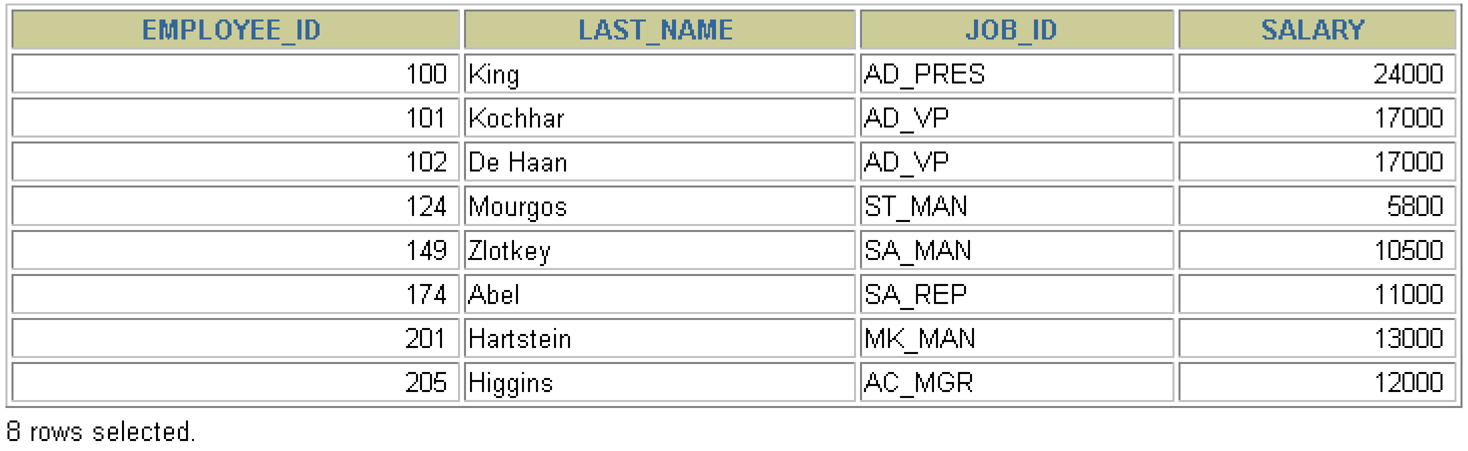


EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

Using the OR Operator

OR requires either conditions to be true.

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

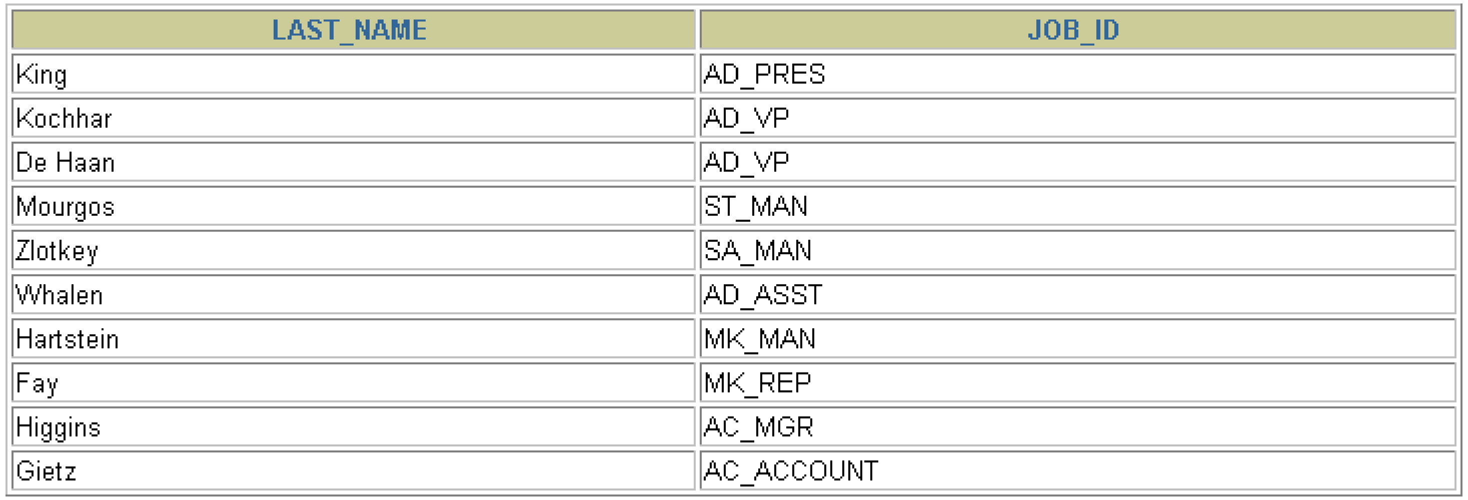


EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

Using the NOT Operator

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



LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

Rules of Precedence

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

Override rules of precedence by using parentheses.

Rules of Precedence

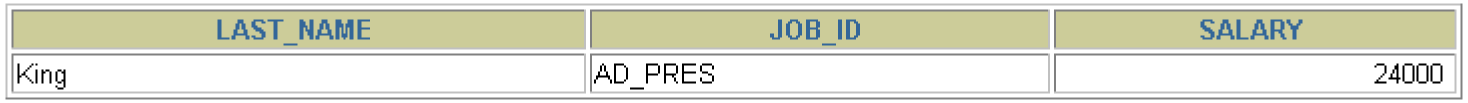
```
SELECT last_name, job_id, salary
FROM   employees
WHERE   job_id = 'SA_REP'
OR      job_id = 'AD_PRES'
AND     salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

Using parenthesis

Use parenthesis to force priority

```
SELECT last_name, job_id, salary
FROM   employees
WHERE     (job_id = 'SA_REP'
OR         job_id = 'AD_PRES')
AND        salary > 15000;
```



LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

Data Manipulation Language

- A DML statement is executed when you:
  - Add new row(s) to a table
    - INSERT
  - Get row(s) from table
    - SELECT
  - Modify existing rows in a table
    - UPDATE
  - Remove existing rows from a table
    - DELETE

Adding a New Row to a Table

70	Public Relations	100	1700

New row

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

…insert a new row into the DEPARMENTS table…

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
70	Public Relations	100	1700

The `INSERT` Statement Syntax

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

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

- Only one row is inserted at a time with this syntax.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

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

- Enclose character and date values within single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO   departments (department_id,
                           department_name    )
VALUES        (30, 'Purchasing');
1 row created.
```

- Explicit method: Specify the `NULL` keyword in the VALUES clause.

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 row created.
```

Inserting Special Values

The `SYSDATE` function records the current date and time.

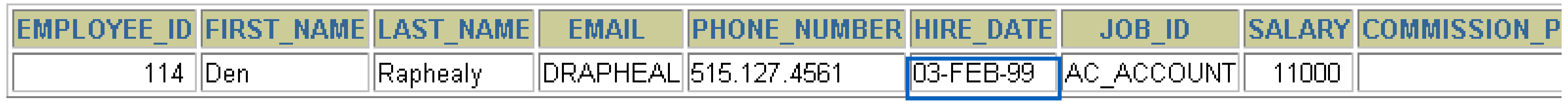
```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES          (113,
                 'Louis', 'Popp',
                 'LPOPP@gmail.com', '515.124.4567',
                 SYSDATE, 'AC_ACCOUNT', 6900,
                 NULL, 205, 100);
1 row created.
```

Inserting Specific Date Values

- Add a new employee.

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

- Verify your addition.



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

Copying Rows from Another Table

- Write your INSERT statement with a

```
INSERT INTO emp
  SELECT * FROM  employees;

4 rows created.
```

- Do not use the VALUES clause.
- It copies all data from employees table into emp table

Copying Specific Columns data from Another Table

```
INSERT INTO emp(id, name, salary)

  SELECT employee_id, last_name, salary
  FROM   employees;

4 rows created.
```

- Match the number of columns in the INSERT clause to those in the subquery.

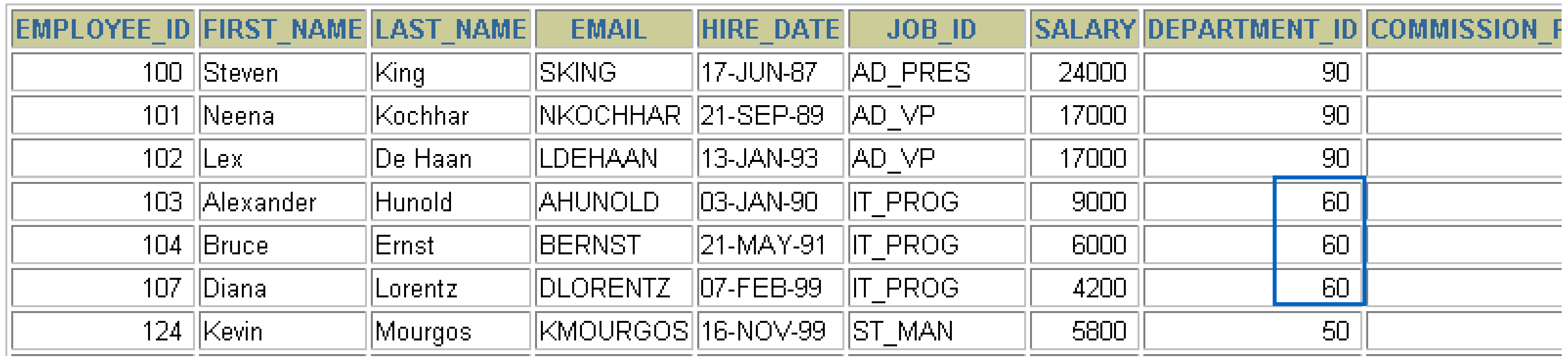
Copying Specific Rows from Another Table

```
INSERT INTO emp_CS(id, name, salary)

  SELECT employee_id, last_name, salary
  FROM   employees;
  WHERE emp_dept = ‘CS’;

4 rows created.
```

- You can filter rows before entering into new table
- It first search out the CS employees and then insert its respective data (id, name, salary) into emp_CS table

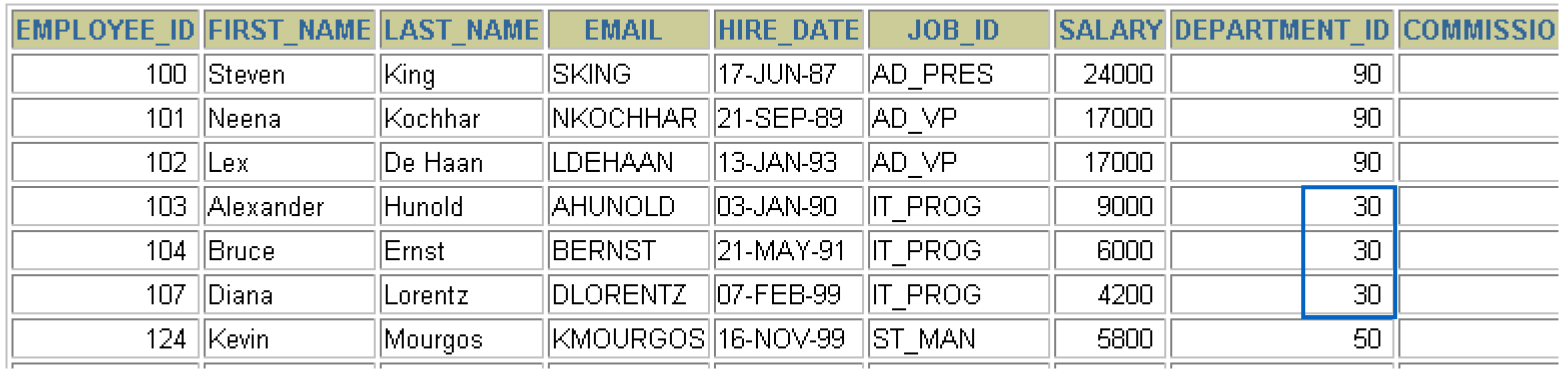
Changing Data in a Table

`EMPLOYEES`



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the `EMPLOYEES` table.



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

The UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement.

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

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

Updating Rows in a Table

- Specific row or rows are modified if you specify the WHERE clause.

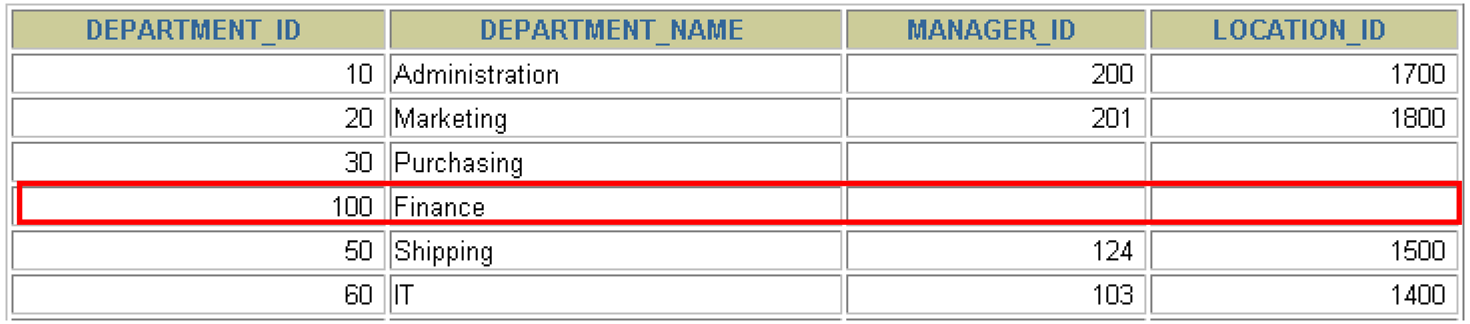
```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause.

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

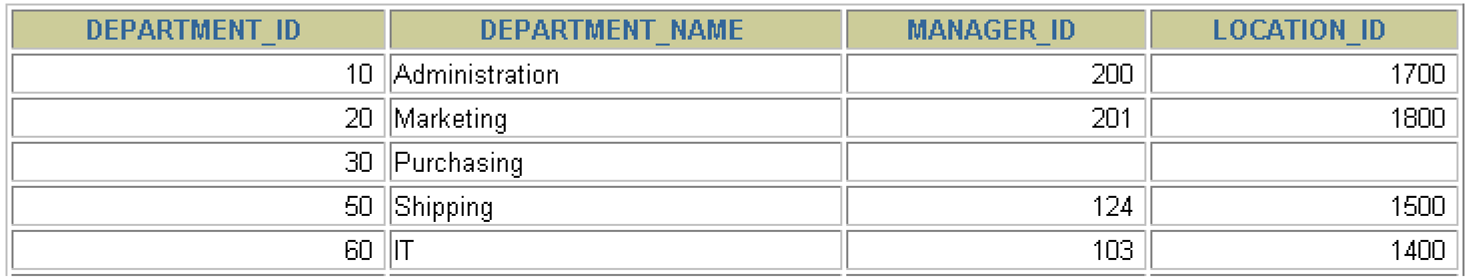
Removing a Row from a Table

DEPARTMENTS



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

Delete a row from the DEPARTMENTS table.



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

The DELETE Statement

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

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

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause.

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

- All rows in the table are deleted if you omit the WHERE claus

```
DELETE FROM  copy_emp;
22 rows deleted.
```

- Deleting a specific row

```
DELETE FROM  copy_emp where empno=10;
1 row deleted.
```