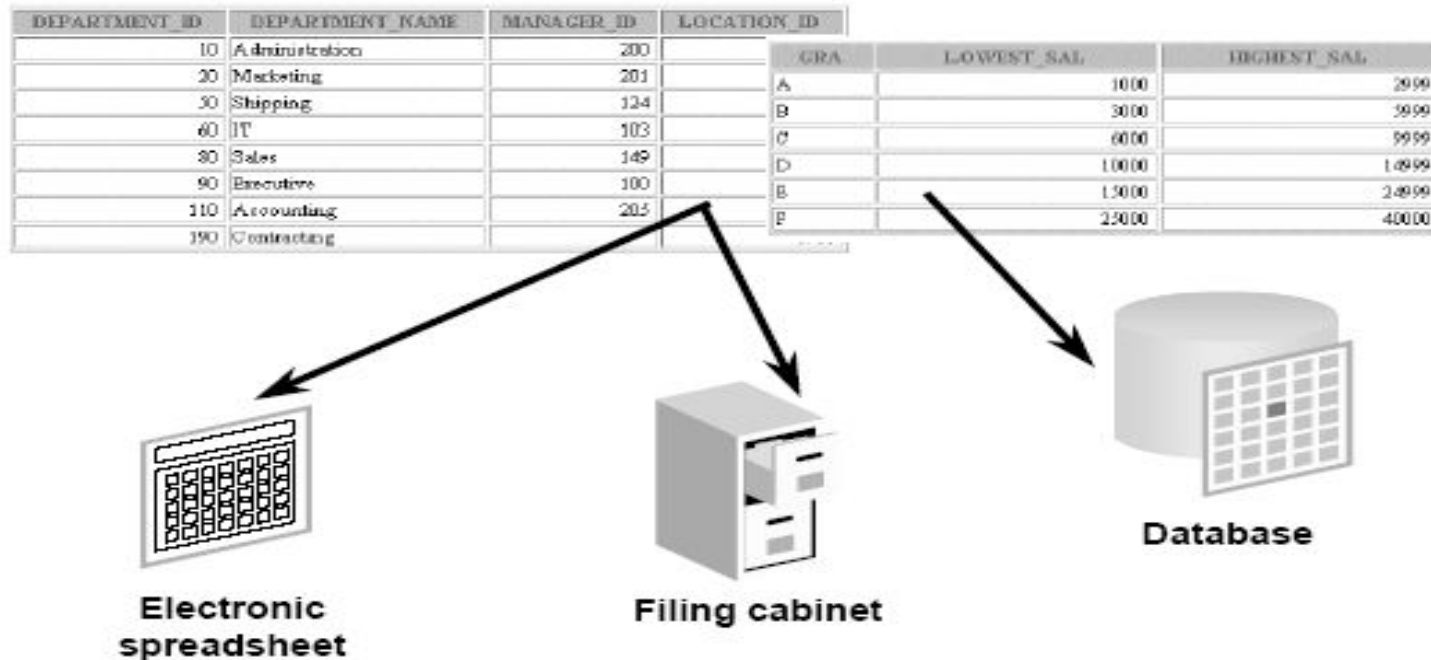


DBMS Concepts and SQL

-BY NIMESH KUMAR DAGUR, CDAC ,
INDIA

Database Concepts



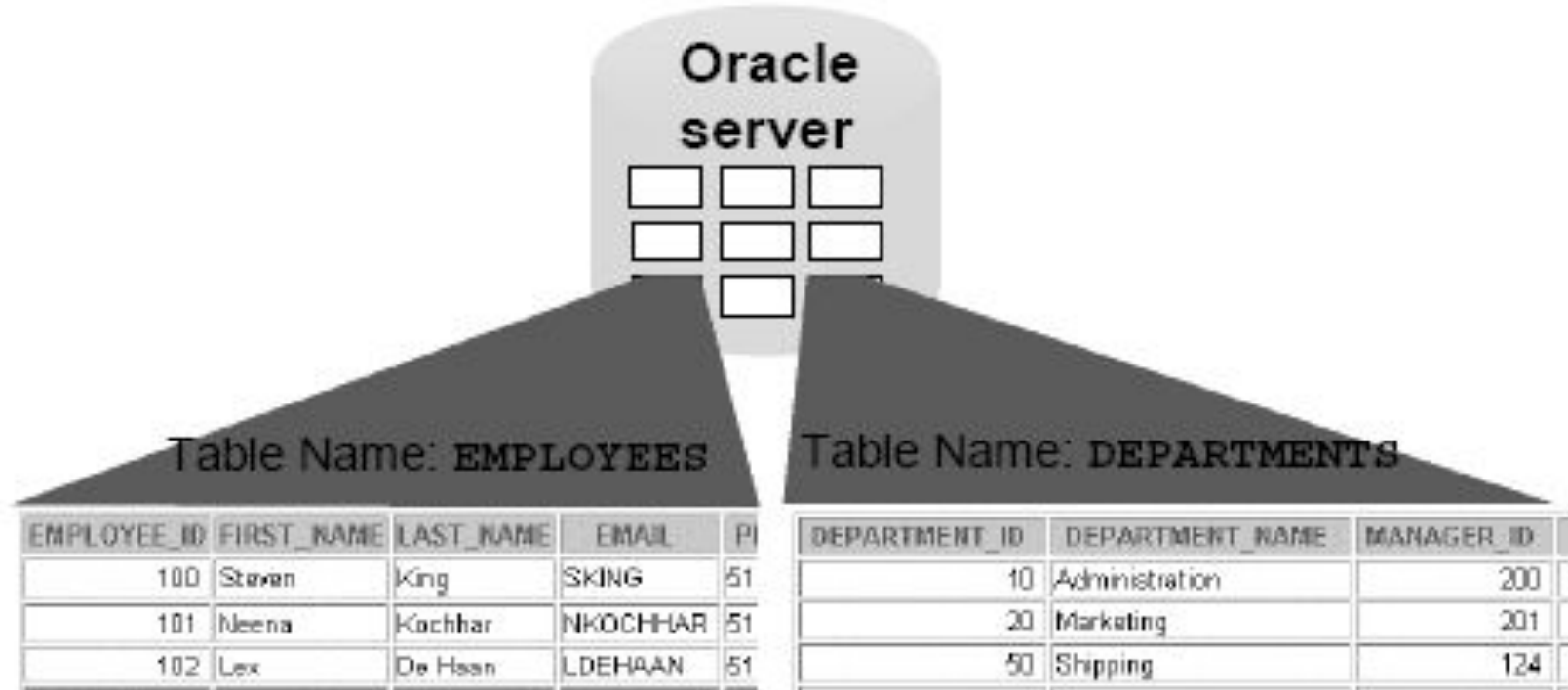
- A **database** is an organized collection of information.
- A DBMS is a program that stores, retrieves, and modifies data in the database on request.

Relational Database Concept

- Dr. E.F. Codd proposed the relational model for database systems in 1970.
- It is the basis for the relational database management system (RDBMS).
- The relational model consists of the following:
 - Collection of objects or relations
 - Set of operators to act on the relations
 - Data integrity for accuracy and consistency

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.



Relating Multiple Tables

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

Table Name: **EMPLOYEES**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lak	De Haan	90
104	Bruce	Ernst	80
202	Pat	Fay	20
205	William	Gietz	110

...



Primary key



Foreign key

Table Name: **DEPARTMENTS**

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



Primary key

Relational Database Terminology

- A single **row** or table representing all data required for a particular employee.
- **A column:** the employee number column is designated as the *primary key*.
- **Data Type**
- A **foreign key** is a column that defines how tables relate to each other.
- **Null value**
- A **field** can be found at the intersection of a row and a column.

2	3	4				
1	EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
	100	King	Steven	24000		90
	101	Kochhar	Neena	17000		90
	102	De Haan	Lex	17000		90
	103	Hunold	Alexander	9000		60
	104	Ernst	Bruce	6000		60
	107	Lorentz	Diana	4200	5	60
	124	Mourgos	Kevin	5900		50
	141	Rajs	Trenna	3500		50
	142	Davies	Curtis	3100		50
	143	Matos	Randall	2600		50
	144	Vargas	Peter	2500		50
	149	Zlotkey	Eleni	10500	.2	60
	174	Abel	Ellen	11000	.3	60
	176	Taylor	Jonathon	8600	.2	60
	178	Grant	Kimberely	7000	.15	
	200	Whalen	Jennifer	4400		10
	201	Hartstein	Michael	13000		20
	202	Fay	Pat	6000		20
	205	Higgins	Shelley	12000		110
	206	Gietz	William	8900		110

Relational Database Properties

A relational database:

- Can be accessed and modified by executing structured query language (SQL) statements
- Contains a collection of tables with no physical pointers
- Uses a set of operators

Communicating with a RDBMS Using SQL

SQL statement
is entered.

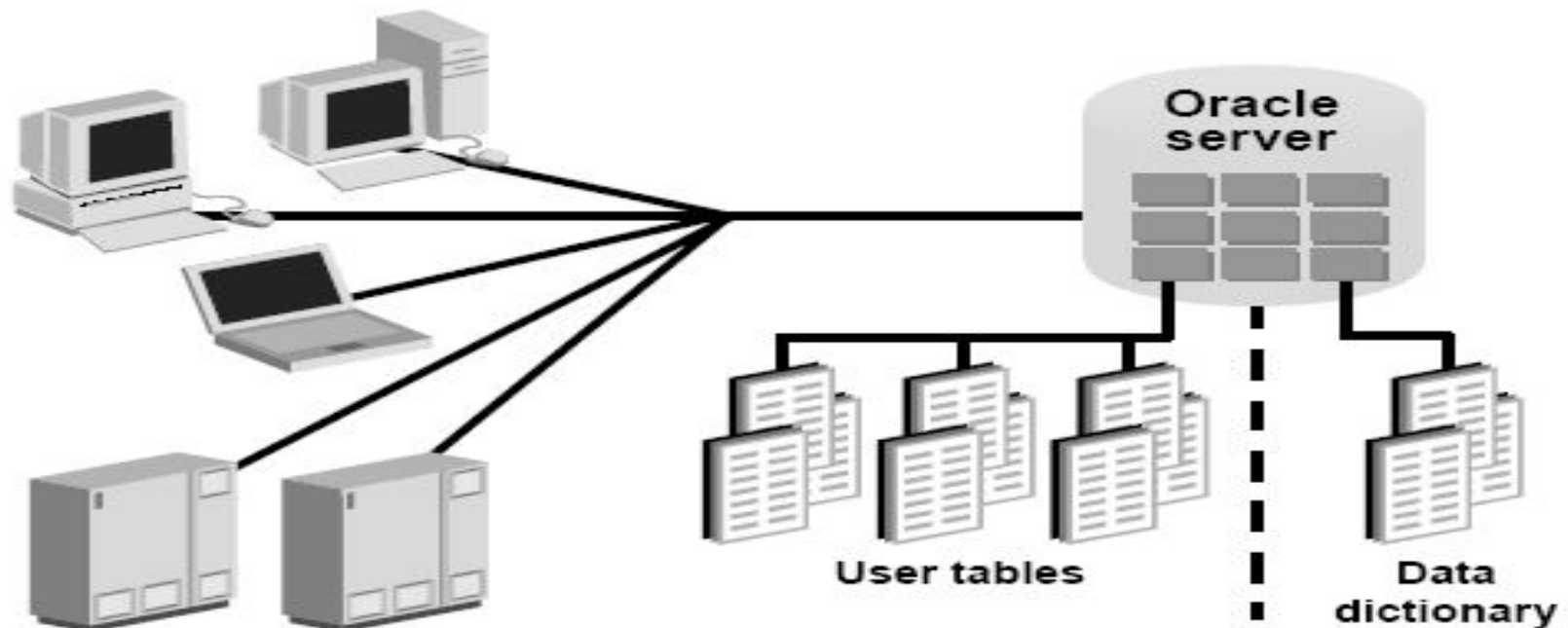
```
SELECT department_name  
FROM departments;
```

Statement is sent to
Oracle Server.

Oracle
server

DEPARTMENT_NAME
Administration
Marketing
Shipping
IT
Sales
Executive
Accounting
Contracting

Relational Database Management System



- Oracle provides a flexible RDBMS called Oracle.
- An Oracle server consists of an Oracle database and an Oracle server instance.
- Every time a database is started, a system global area (SGA) is allocated, and Oracle background processes are started.

SQL Statements

- **SQL** is the language used to communicate with the server to access, manipulate, and control data.
- Oracle SQL complies with industry-accepted standards.
- Industry-accepted committees are the American National Standards Institute (**ANSI**) and the International Standards Organization (**ISO**).
- Both ANSI and ISO have accepted SQL as the standard language for relational databases.

SQL Statements

SELECT	Data retrieval
INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)

SQL Statements

Statement	Description
SELECT	Retrieves data from the database
INSERT UPDATE DELETE MERGE	Enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as <i>data manipulation language</i> (DML).
CREATE ALTER DROP RENAME TRUNCATE	Sets up, changes, and removes data structures from tables. Collectively known as <i>data definition language</i> (DDL).
COMMIT ROLLBACK SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
GRANT REVOKE	Gives or removes access rights to both the Oracle database and the structures within it. Collectively known as <i>data control language</i> (DCL).

Naming Rules

Table names and column names:

- Must begin with a letter
- Must be 1–30 characters long
- Must contain only **A–Z**, **a–z**, **0–9**, **_**, **\$**, and **#**
- Must not duplicate the name of another object owned by the same user
- Must not be an Oracle server reserved word

Writing SQL Statements

- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.

Displaying Table Structure

- Use DESCRIBE command to display the structure of a table.

```
DESC[RIBE] tablename
```

Constraints

- Constraints enforce rules on data stored.
- The Oracle Server uses constraints to prevent invalid data entry into tables.
- Constraints are used for:
 - Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table.
 - Prevent the deletion of a table if there are dependencies from other tables

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

Constraint Guidelines

- Create a constraint either:
 - At the same time as the table is created, or
 - After the table has been created
- Define a constraint at the column or table level.

Defining Constraints

- **Constraints can be defined at one of two levels.**


Constraint Level	Description
Column	References a single column and is defined within a specification for the owning column; can define any type of integrity constraint
Table	References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except NOT NULL

The PRIMARY KEY Constraint

- The PRIMARY KEY constraint is a column or set of columns that uniquely identifies each row in a table.
- A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table.
- This constraint enforces uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

The PRIMARY 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

...

Not allowed
(Null value)



INSERT INTO

	Public Accounting		1400
50	Finance	124	1500

Not allowed
(50 already exists)



The PRIMARY KEY Constraint

- PRIMARY KEY constraints can be defined at the column level or table level.
- A table can have only one PRIMARY KEY constraint but can have several UNIQUE constraints.

The PRIMARY KEY Constraint Defined at Column Level

- *CREATE TABLE CUST_MSTR (
CUST_NO VARCHAR2(10),
FNAME VARCHAR2(25),
MNAME VARCHAR2(25),
LNAME VARCHAR2(25),
DOB_INC DATE,
OCCUP VARCHAR2(25),
PHOTOGRAPH VARCHAR2(25),
SIGNATURE VARCHAR2(25),
PANCOPY VARCHAR2(1),
FORM60 VARCHAR2(1), PRIMARY KEY(Cust_no)
);*

The PRIMARY KEY Constraint Defined at Table Level

```
CREATE TABLE FD_MSTR(  FD_SER_NO VARCHAR2(10),
  SF_NO VARCHAR2(10),
  BRANCH_NO VARCHAR2(10),INTRO_CUST_NO
  VARCHAR2(10),
  INTRO_ACCT_NO VARCHAR2(10),INTRO_SIGN VARCHAR2(1),
  ACCT_NO VARCHAR2(10),TITLE VARCHAR2(30),
  CORP_CUST_NO VARCHAR2(10),CORP_CNST_TYPE
  VARCHAR(4),
  VERI_EMP_NO VARCHAR2(10),VERI_SIGN VARCHAR2(1),
  MANAGER_SIGN VARCHAR2(1),
  PRIMARY KEY(FD_SER_NO, CORP_CUST_NO));
```


The FOREIGN KEY Constraint

- Foreign key represent relationships between tables.
- A Foreign key is a column (or a group of columns) whose values are derived from the Primary key or unique key of other table
- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, not physical, pointers.
- FOREIGN KEY constraints can be defined at the column or table constraint level.
- The foreign key is defined in the child table, and the table containing the referenced column is the parent table.

The FOREIGN KEY Constraint

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1600
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

PRIMARY
KEY →

...



EMPLOYEES

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

← FOREIGN
KEY

...



INSERT INTO

200	Ford	9
201	Ford	60

Not allowed
(9 does not
exist)

← Allowed

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 FOREIGN KEY Constraint Defined at the Column Level

- CREATE TABLE EMP_MSTR(EMP_NO VARCHAR2(10)
PRIMARY KEY,
BRANCH_NO VARCHAR2(10) REFERENCES BRANCH_MSTR,
FNAME VARCHAR2(25),MNAME VARCHAR2(25),
LNAME VARCHAR2(25),DEPT VARCHAR2(30),
DESIG VARCHAR2(30));

The FOREIGN KEY Constraint Defined at the Table Level

- `CREATE TABLE ACCT_FD_CUST_DTLS(ACCT_FD_NO
VARCHAR2(10),
CUST_NO VARCHAR2(10),
FOREIGN KEY (CUST_NO) REFERENCES
CUST_MSTR(CUST_NO));`

The FOREIGN KEY Constraint Defined with ON DELETE CASCADE

The FOREIGN KEY Constraint Defined with ON DELETE SET NULL

The UNIQUE Constraint

- A UNIQUE key integrity constraint requires that every value in a column or set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or set of columns.
- The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*.
- If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.
- UNIQUE constraints allow the multiple entries of NULL into the column


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

...

UNIQUE constraint





INSERT INTO



208	Smith	JSMITH
209	Smith	JSMITH

Allowed

Not allowed:
already exists



UNIQUE constraints defined at the column level

- CREATE TABLE CUST_MSTR2 (
CUST_NO VARCHAR2(10) UNIQUE,
FNAME VARCHAR2(25),
MNAME VARCHAR2(25),
LNAME VARCHAR2(25),
DOB_INC DATE,
OCCUP VARCHAR2(25),
PHOTOGRAPH VARCHAR2(25),
SIGNATURE VARCHAR2(25),
PANCOPY VARCHAR2(1),
FORM60 VARCHAR2(1));

UNIQUE constraints defined at the Table level

- CREATE TABLE CUST_MSTR3 (
CUST_NO VARCHAR2(10),
FNAME VARCHAR2(25),
MNAME VARCHAR2(25),
LNAME VARCHAR2(25),
DOB_INC DATE,
OCCUP VARCHAR2(25),
PHOTOGRAPH VARCHAR2(25),
SIGNATURE VARCHAR2(25),
PANCOPY VARCHAR2(1),
FORM60 VARCHAR2(1),
UNIQUE(CUST_NO));

The NOT NULL Constraint

- The NOT NULL constraint ensures that the column contains no null values.
- Columns without the NOT NULL constraint can contain null values by default.
- The NOT NULL constraint can be specified only at the column level, not at the table level.

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	80
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	80
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**



**Absence of NOT NULL
constraint**
(Any row can contain
null for this column.)

The NOT NULL Constraint

- CREATE TABLE CUST_MSTR4(
CUST_NO VARCHAR2(10),
FNAME VARCHAR2(25),
MNAME VARCHAR2(25),
LNAME VARCHAR2(25),
DOB_INC DATE NOT NULL,
OCCUP VARCHAR2(25),
PHOTOGRAPH VARCHAR2(25),
SIGNATURE VARCHAR2(25),
PANCOPY VARCHAR2(1),
FORM60 VARCHAR2(1));

Viewing Constraints

- Query the USER_CONSTRAINTS table to view all constraint definitions and names.

```
SELECT constraint_name, constraint_type,  
       search_condition FROM user_constraints;
```

- Constraints that are not named by the table owner receive the system-assigned constraint name. In constraint type,

C stands for CHECK,

P for PRIMARY KEY,

R for referential integrity, and

U for UNIQUE key.

Notice that the NOT NULL constraint is really a CHECK constraint.

Viewing the Columns Associated with Constraints

- View the columns associated with the constraint names in the USER_CONS_COLUMNS view.

```
SELECT constraint_name, column_name FROM user_cons_columns;
```


SQL Operators & Functions

Arithmetic Expressions

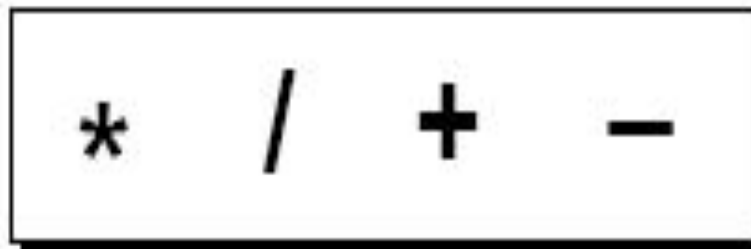
Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

- Create expressions with number and date data by using arithmetic operators.

Example:

1. ***select ENAME, SAL + 100, JOB from emp where DEPTNO = 20;***
2. ***select sysdate + 2 from dual;***

Operator 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
Ernet	6000	72100

...

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

20 rows selected.

Using Parentheses

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

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	288000
Kochhar	17000	204000
De Haan	17000	204000
Hunold	9000	108000
Ernst	6000	72000

...

Hartstein	13000	156000
Fay	6000	72000
Higgins	12000	144000
Gietz	8300	100800

20 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.

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
Malos	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	2800
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.

Logical Conditions

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are true
OR	Returns TRUE if <i>either</i> 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	Harstein	MK_MAN	13000

Using the OR Operator

OR requires either 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
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5900
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected

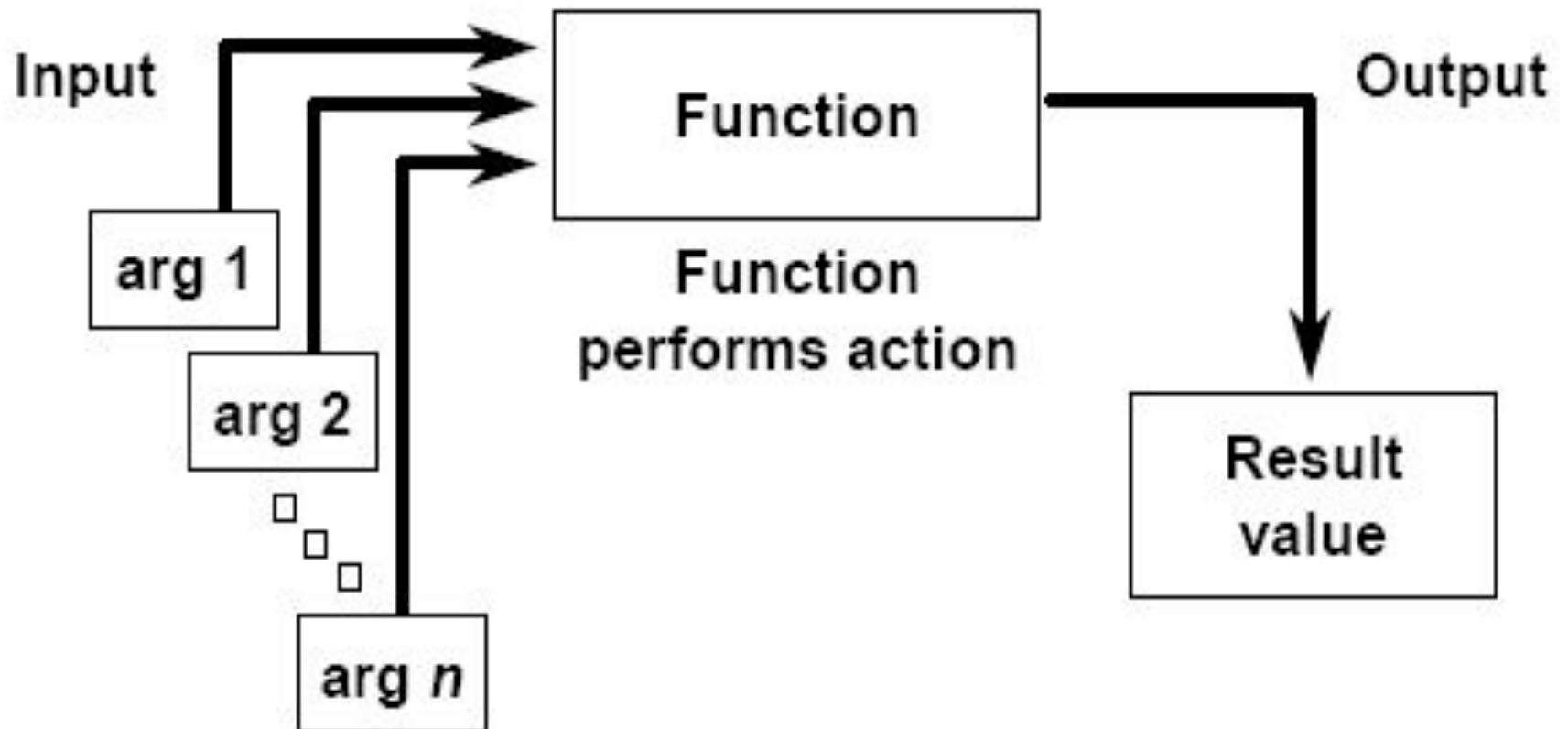
Using the NOT Operator

```
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

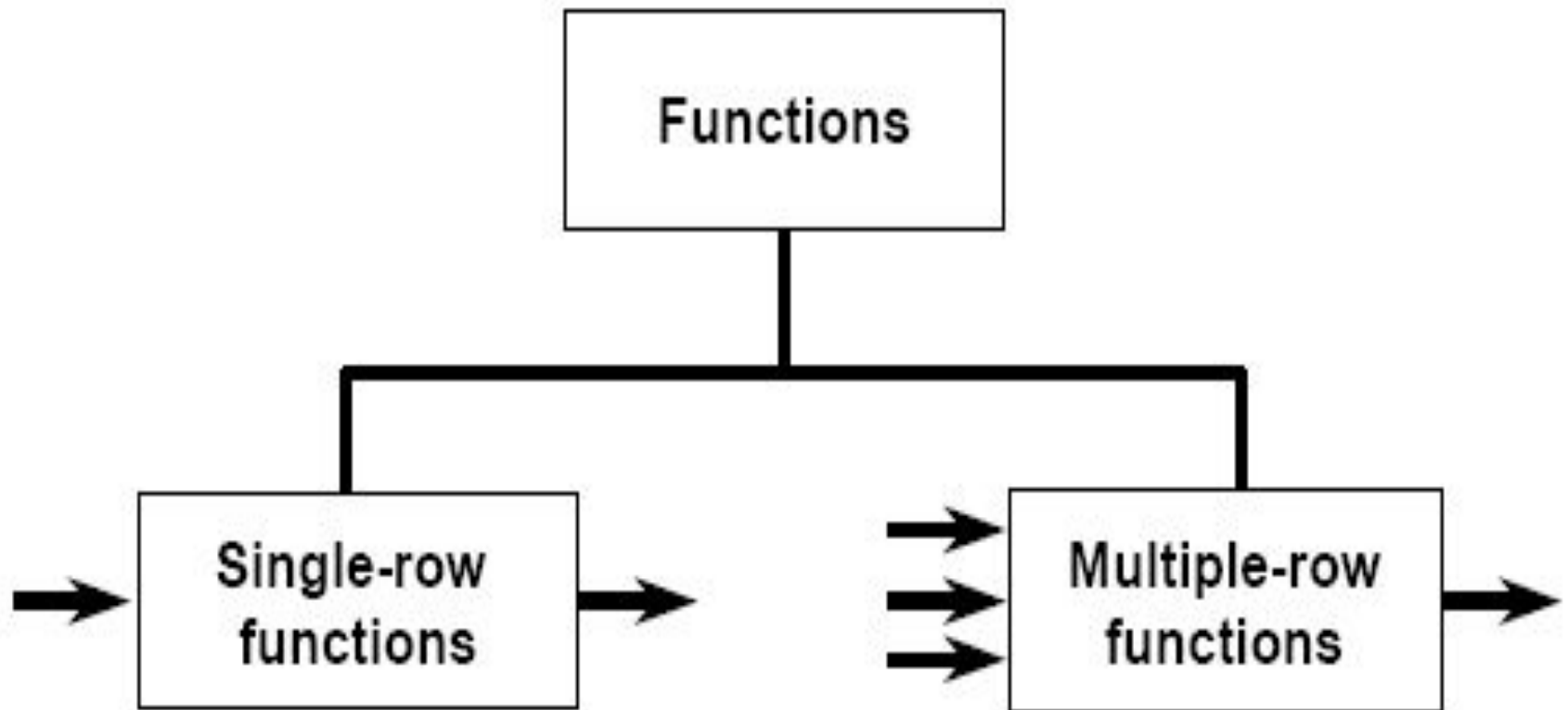
SQL Functions



SQL Functions

- Functions are a very powerful feature of SQL and can be used to do the following:
 - Perform calculations on data
 - Modify individual data items
 - Manipulate output for groups of rows
 - Format dates and numbers for display
 - Convert column data types

Two Types of SQL Functions



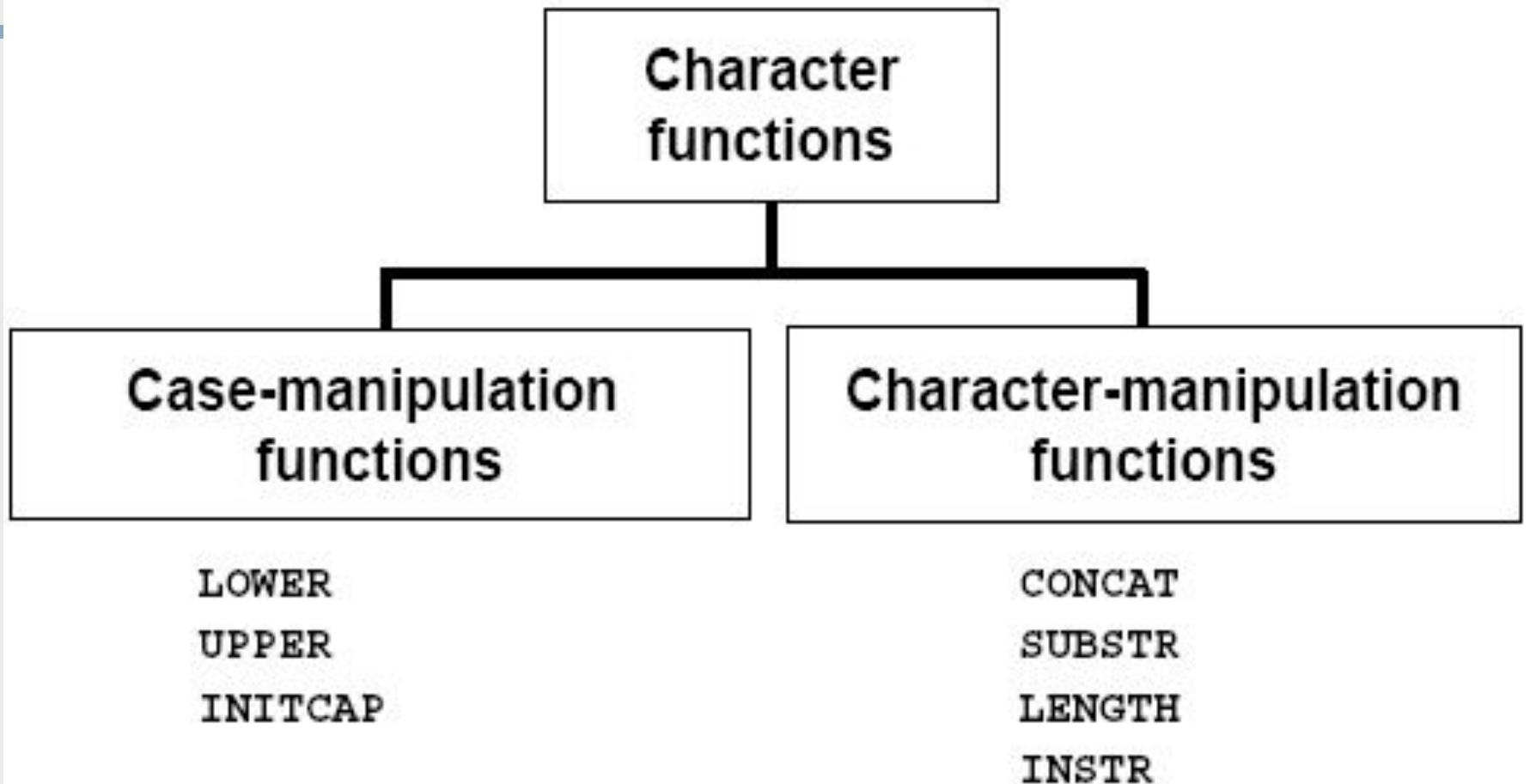
Single-Row Functions

- These functions operate on single rows only and return one result per row.
- There are different types of single-row functions. :
 - **Character(String):** Accept character input and can return both character and number values.
 - **Number:** Accept numeric input and return numeric values
 - **Date:** Operate on values of the DATE data type
 - **Conversion:** Convert a value from one data type to another

Multiple-Row Functions

- Functions can manipulate groups of rows to give one result per group of rows. These functions are known as **group functions**.

Character (String) Functions



Function	Purpose
LENGTH (<i>column</i>)	Returns the number of characters in the expression
INSTR (<i>column</i> , ' <i>string</i> ', [<i>m</i>], [<i>n</i>])	Returns the numeric position of a named string. Optionally, you can provide a position <i>m</i> to start searching, and the occurrence <i>n</i> of the string. <i>m</i> and <i>n</i> default to 1, meaning start the search at the beginning of the search and report the first occurrence.

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
<code>CONCAT('Hello', 'World')</code>	HelloWorld
<code>SUBSTR('HelloWorld',1,5)</code>	Hello
<code>LENGTH('HelloWorld')</code>	10
<code>INSTR('HelloWorld', 'W')</code>	6

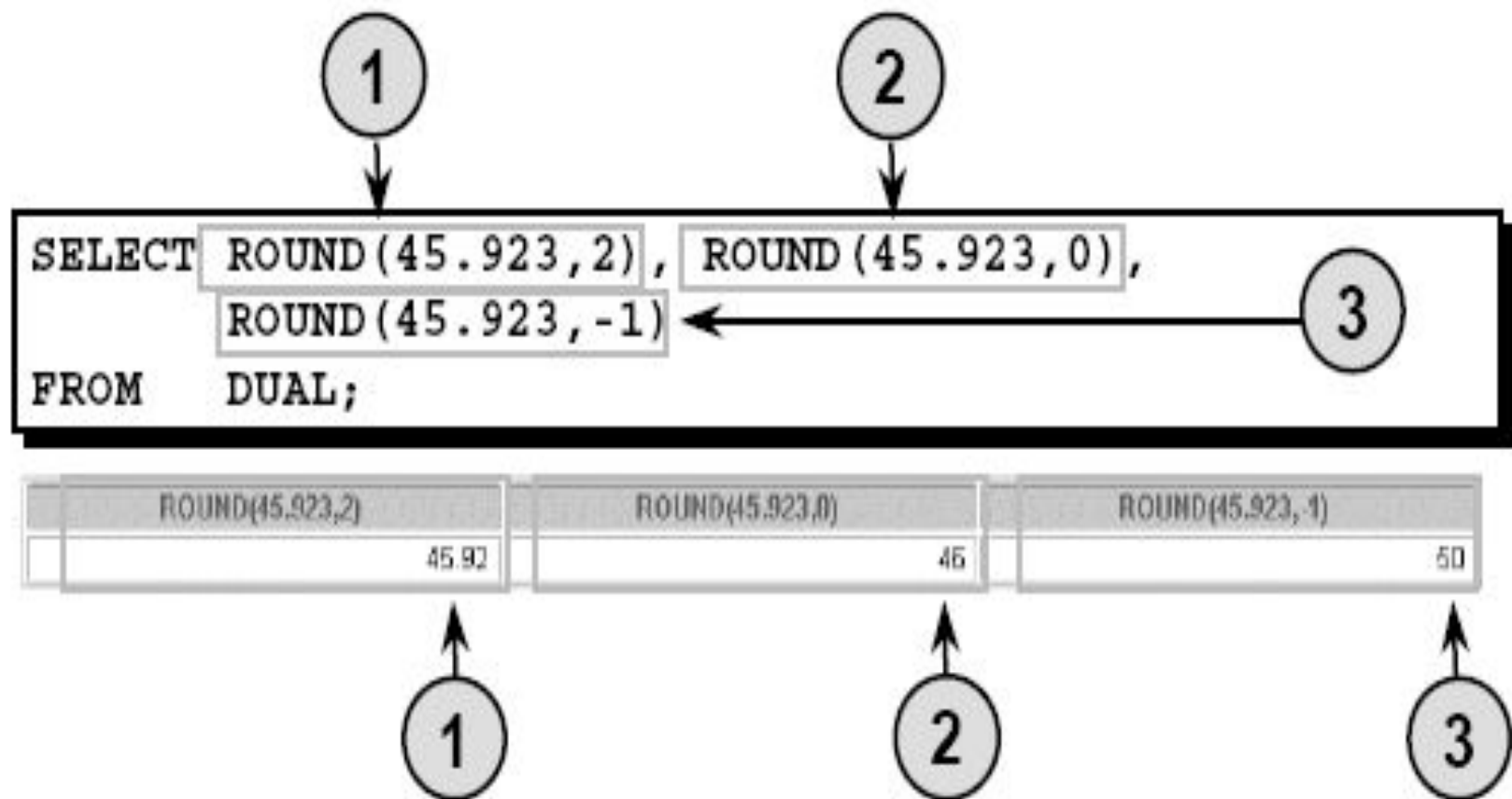
Case Manipulation Functions

- LOWER, UPPER, and INITCAP are the three case-conversion functions.
- **LOWER:** Converts mixed case or uppercase character strings to lowercase
- **UPPER:** Converts mixed case or lowercase character strings to uppercase
- **INITCAP:** Converts the first letter of each word to uppercase and remaining letters to lowercase

Number Functions

- **ABS(*n*)**: Return the absolute value of '*n*'
- **POWER(*m*,*n*)**: return *m* raised to the *n*th power
- **ROUND(*column|expression*, *n*)**: Rounds the column, expression, or value to *n* decimal places, or, if *n* is omitted, no decimal places. (If *n* is negative, numbers to left of the decimal point are rounded.)
- **TRUNC(*column|expression*, *n*)**: Truncates the column, expression, or value to *n* decimal places, or, if *n* is omitted, then *n* defaults to zero
- **MOD(*m*,*n*)**: Returns the remainder of *m* divided by *n*

Using the ROUND Function



DUAL is a dummy table you can use to view results from functions and calculations.

Using the TRUNC Function

Diagram illustrating the SQL query using the TRUNC function:

```
SELECT TRUNC(45.923, 2), TRUNC(45.923),  
       TRUNC(45.923, -2)  
FROM DUAL;
```

Annotations:

- 1 points to the first argument (45.923) of the first TRUNC function.
- 2 points to the second argument (2) of the first TRUNC function.
- 3 points to the third TRUNC function (TRUNC(45.923, -2)).

TRUNC(45.923, 2)	TRUNC(45.923)	TRUNC(45.923, -2)
45.92	45	0

Annotations:

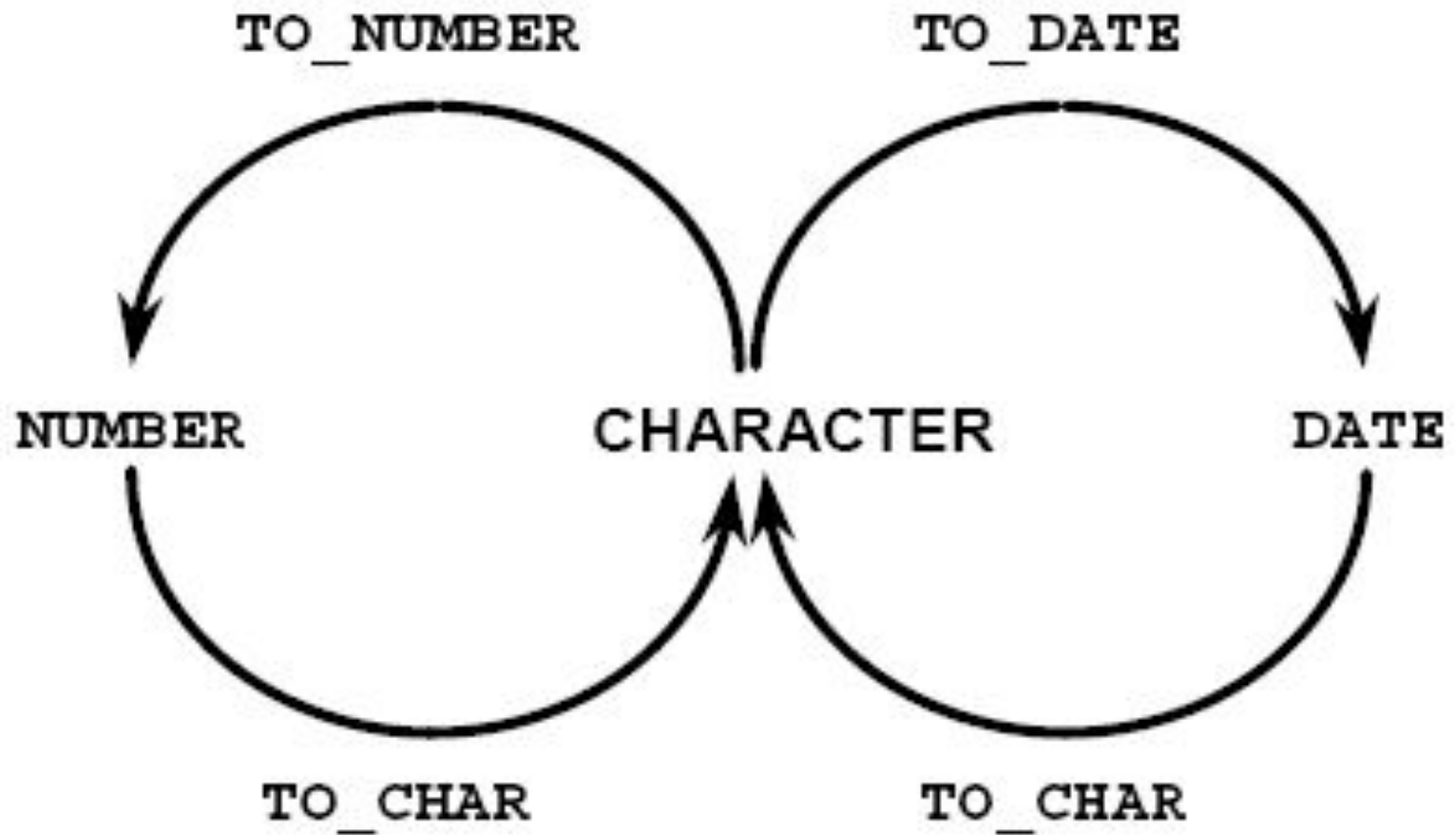
- 1 points to the result 45.92.
- 2 points to the result 45.
- 3 points to the result 0.

Using the MOD Function

MOD: Returns remainder of division

`MOD(1600, 300)`  100

Conversion Functions



Conversion Functions (contd..)

Function	Purpose
TO_CHAR (<i>number date</i> , [<i>fmt</i>])	Converts a number or date value to a VARCHAR2 character string with format model <i>fmt</i> .
TO_NUMBER (<i>char</i> , [<i>fmt</i>])	Converts a character string containing digits to a number in the format specified by the optional format model <i>fmt</i> .
TO_DATE (<i>char</i> , [<i>fmt</i>])	Converts a character string representing a date to a date value according to the <i>fmt</i> specified. If <i>fmt</i> is omitted, the format is DD-MON-YY.

Example

- SQL> select to_char(123) from dual;
- SQL> select to_char(sysdate, 'dd/mm/yyyy') from dual;
- SQL> select to_date('22/08', 'dd-yy') from dual;
- SQL> select to_Number('1233') from dual;

Group Functions

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

EMPLOYEES

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

...

20 rows selected.

The maximum
salary in
the EMPLOYEES
table.

MAX(SALARY)
24000

Types of Group Functions

- **AVG:** Average value of n , ignoring null values
- **COUNT (*expr*) :** Number of rows, where *expr* is not null
- **COUNT (*) :** Number of rows, including duplicates and rows with nulls
- **MAX:** Maximum value of *expr*, ignoring null values
- **MIN:** Minimum value of *expr*, ignoring null values
- **SUM:** Sum values of n , ignoring null values

You can use AVG and SUM for numeric data.

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

Displaying Data from Multiple Tables

Obtaining Data from Multiple Tables

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1900
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
150	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Cartesian Products

- A Cartesian product is formed when:
 - A join condition is omitted
 - A join condition is invalid
 - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a `WHERE` clause.

Generating a Cartesian Product

EMPLOYEES (20 rows)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

**Cartesian
product:** →
20x8=160 rows

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700

160 rows selected.

Types of Joins

There are multiple ways to join tables.

- Equijoins
- Non-equijoins
- Outer joins
- Self joins
- Cross joins
- Natural joins
- Full or outer joins

Joining Tables Using Oracle Syntax

Use a join to query data from more than one table.

```
SELECT    table1.column, table2.column  
FROM      table1, table2  
WHERE     table1.column1 = table2.column2;
```

- Write the join condition in the WHERE clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

EMPLOYEES

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

...



Foreign key

DEPARTMENTS

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

...



Primary key

Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,  
       employees.department_id, departments.department_id,  
       departments.location_id  
FROM   employees, departments  
WHERE  employees.department_id = departments.department_id;
```

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

...

19 rows selected.

Using Table Aliases

- Simplify queries by using table aliases.
- Improve performance by using table prefixes.

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

Guidelines

- Table aliases can be up to 30 characters in length, but shorter is better.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must
- be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid only for the current SELECT statement.

Joining More than Two Tables

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Yargas	50
Zlotkey	80
Abel	80
Taylor	80

DEPARTMENTS

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

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

20 rows selected.

- To join n tables together, you need a minimum of $n-1$ join conditions. For example, to join three tables, a minimum of two joins is required.

Example

```
SELECT e.last_name, d.department_name, l.city  
FROM employees e, departments d, locations l  
WHERE e.department_id = d.department_id  
AND d.location_id = l.location_id;
```

Non-Equijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8500

...
20 rows selected.

JOB_GRADES

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



Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

- A non-equijoin is a join condition containing something other than an equality operator
- The relationship is obtained using an operator other than equals (=).

Retrieving Records with Non-Equi Joins

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

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5000	D
Rajs	3500	B
Davis	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

20 rows selected.

Outer Joins

DEPARTMENTS

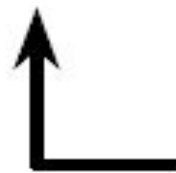
DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

0 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...
20 rows selected.



There are no employees in department 190.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row will not appear in the query result

Outer Joins Syntax

- You use an outer join to also see rows that do not meet the join condition.
- The Outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column  
FROM   table1, table2  
WHERE  table1.column(+) = table2.column;
```

```
SELECT table1.column, table2.column  
FROM   table1, table2  
WHERE  table1.column = table2.column(+);
```

Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Raj	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
...		
Gietz	110	Accounting
		Contracting

20 rows selected.

Self Joins

EMPLOYEES (WORKER)

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

EMPLOYEES (MANAGER)

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

...

...



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

Joining a Table to Itself

Sometimes you need to join a table to itself

Joining a Table to Itself

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id;
```

WORKER.LAST_NAME 'WORKSFOR' MANAGER.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold

...

19 rows selected.