FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

DISEÑO DE BASES DE DATOS Y SQL

MATERIAL DIDACTICO

ING. JORGE BARBA ATILANO

MARZO 1995

Class Notes

March 1988

SQL*Plus

Jorg Barba Atilano

ORACLE

SQL*Plus Class Notes

Contributing Authors: Nimish Mehta, Chris Schock, Mark Rosen, Judith Vandenberg
Reviewers: Glynn Durham, Martin Gardner, Brad Goodwin, Tim Kelley, Carol Zimmerman

Table of Contents

Chapter 1 - Introduction

Chapter 2 - Database Queries

Pract

Chapter 3 - Data Manipulation

Pract.

Chapter 4 - Creating Tables & Views

Práctica

Chapter 5 - SQL*Plus Reporting

Práctica

Chapter 6 - Functions

Practice

Chapter 7 - Advanced Queries

Práctica

Exercises and Answers

Tables

1. Introduction

Traditional Problems Accessing Information

- Data vs Information
- A Typical Question:

> *"Which departments in our corporation have payrolls exceeding $2,200 per month per employee?"*

- Common problems getting this answered:

TURNAROUND TOO LONG

MIS backlog

COSTS TOO MUCH

programming time

INCONVENIENCE

requestor must justify priority

Purpose of

*SQL*Plus Course*

- **SQL*Plus: A way to bypass traditional problems**
  - Flexible
  - Quick Turnaround of queries
  - Easy to learn, easy to use
  - Reliable
- **You learn**
  - To create and update tables and views
  - To ask your database almost anything
  - To format professional reports

What is a Database?

- A database is an *integrated* collection of data
- Every person and every program authorized to access it can do so
- The data can be modified by those authorized to do so
- A database properly designed should *minimize* the amount of redundant information

What is a DBMS?

A ***DBMS*** (DataBase Management System)
is a software program that

- stores, retrieves, and modifies data
- guards the data's consistency
- solves concurrency problems
- allows a universal interface to the data
- regulates access to the data

3 Generations of DBMSs

- ***HIERARCHICAL*** ***1960s***
  a programmer can only specify one-parent-multiple-children relationships

- ***NETWORK*** ***1970s***
  a programmer can specify specific multiple-parents-multiple-children relationships

- ***RELATIONAL*** ***1980s +***
  the system automatically establishes all possible relationships

  originated with IBM's ALPHA model (EF Codd, 1970)

  ORACLE V2 - 1st commercial RDBMS (1979)

Features of an RDBMS

- Table-Like Representation of Data
- 4th Generation Language (*4GL*)
  - *Nonprocedural* Syntax
  - English-like (eg, SQL)
- Full Relational Capabilities
  - Automatic Navigation
  - All *Relational* Operators
- Flexibility
  - Easily modify data
  - Easily change database structure
- Integrated *Data Dictionary*

SQL's History

- **IBM's San Jose Research (1970s)**

  several early SQL predecessors

- **SQL published in 1976**

  Oracle announces first commercial RDBMS based on SQL (1979)

  SQL/DS: IBM's first RDBMS (1982)

  DB2: IBM's second RDBMS (1985)

- **ANSI makes SQL the Industry Standard**

  (Oct. 1986)

The Relational Approach

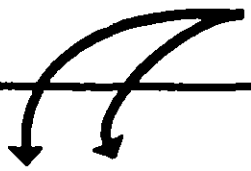
- In a relational database system information is organized in the form of tables.
- Categories of information are listed across the top of each table.
- Individual cases are listed down the left side.
- In this form you can readily visualize, understand and use this information.

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1,400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30

Tables, Rows and Columns

- Each column contains one kind of information.
- Each row is made up of columns that hold one value. For example, the SAL column in the row for JONES has the value 2,975.



EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1,400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30

Related Tables

- The information in one table may be related to the information in another.
- Each employee in ***EMP***, for example, has a department number (***DEPTNO***) that refers to a department number in a table named ***DEPT***.

DEPT Table

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON
30	SALES	CHICAGO
20	RESEARCH	DALLAS
10	ACCOUNTING	NEW YORK

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1,400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30

- **The ability to relate information in one table to another allows you to organize information in separate, manageable units.**

- You can keep employee information logically distinct from department information by storing it in a separate table.

- You may join information from two or more tables if you wish: for example, to find the location (*LOC*) in *DEPT* of any employee in *EMP*.

- When tables have related columns the common values in these columns allow join operations.

*SQL*Plus Commands Not Found in SQL*

- In addition to standard *SQL* commands, Oracle's *SQL*Plus* language includes additional commands called *SQL*Plus* commands.
- These commands do not get stored in the *SQL* buffer.
- *SQL*Plus* commands are used to customize sophisticated reports, edit *SQL* statements, provide a help facility and maintain system variables.
- *SQL*Plus* commands:

@	define	pause
#	del	quit
$	describe	remark
/	disconnect	run
accept	document	save
append	edit	set
break	exit	show
btitle	get	spool
change	help	sqlplus
clear	host	start
column	input	timing
compute	list	ttitle
connect	newpage	undefine
copy		

SQL Commands

- *SQL* commands are used to create, store, change, retrieve and maintain the information in an Oracle Database.

- A *SQL* command is stored in a part of memory called the SQL buffer where it remains until a new command is entered.

- *SQL* has many keywords. The 17 which begin statements are:

alter	drop	revoke
audit	grant	rollback*
commit*	insert	select
comment	lock	update
create	no audit	validate
delete	rename	

*** While part of *SQL*, these commands do not require a semi-colon nor do they get stored in the *SQL* buffer.**

2. Database Queries

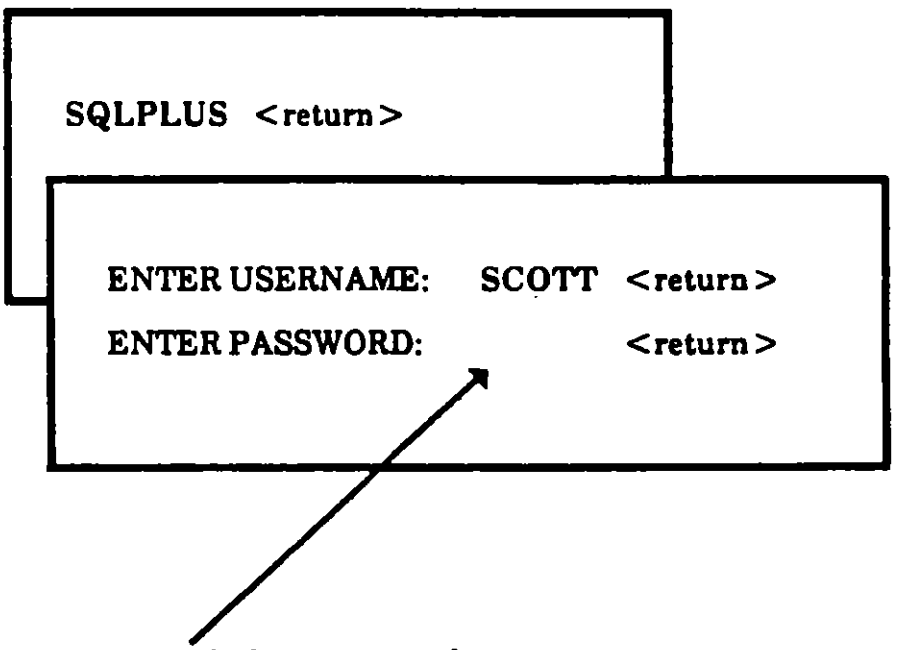
*Logging onto SQL*Plus*

- Access your operating system

  USERID:

  PASSWORD:

- At the Operating System prompt:



- Password does not show

Getting HELP

- List of all *SQL* and *SQL*Plus* commands:

```
SQL>    HELP
```

- Information about a particular command:

```
SQL>    HELP SELECT

SQL>    HELP CREATE
```

- To see a list of the current help topics:

```
SQL>    HELP TOPICS
```

The DATA DICTIONARY

- The ***DATA DICTIONARY*** is a group of tables and views that contains descriptive information about

  Tables

  User access privileges

  Other features of the database

- These are the most frequently used tables when accessing the data dictionary:

  TAB
  A list of the tables, views, and synonyms that you have created

  DTAB
  Tables that make up the data dictionary

  COL
  A list of all column definitions for the tables you created

  CATALOG (CAT) (USER_TABLES)
  A list of all tables you have access to

  CAT

The DESCRIBE Command

- If you want to obtain a description of a specific table, use the ***DESCRIBE*** command

- The description will contain

NAME
name of column

NULL?
whether null values are allowed in this column

Type
char(w)
numeric(w,d)
date
raw

w = width
d = decimal places

```
SQL> DESCRIBE DEPT
```

Name	NULL?	Type
DEPTNO		NUMBER(2)
DNAME		CHAR(14)
LOC		CHAR(13)

General Query Syntax

- By using *SQL* you can query a database in an endless number of ways

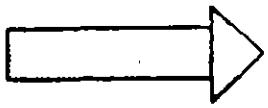
- Although *SQL* is highly flexible,it still has a very specific syntax

- It is possible to phrase a *SQL* question wrong, and end up with either no results at all, or answers to a question you didn't realize you asked

- The overall syntax is very simple:

```
    SELECT ...
    FROM ...
  [ WHERE ...    ]
[   GROUP BY ...   ]
[ [ HAVING ...   ] ]
  [ ORDER BY ... ]
```

Selecting All Columns



What data are in the EMP table?

```
SQL> SELECT *
     FROM   EMP;
```

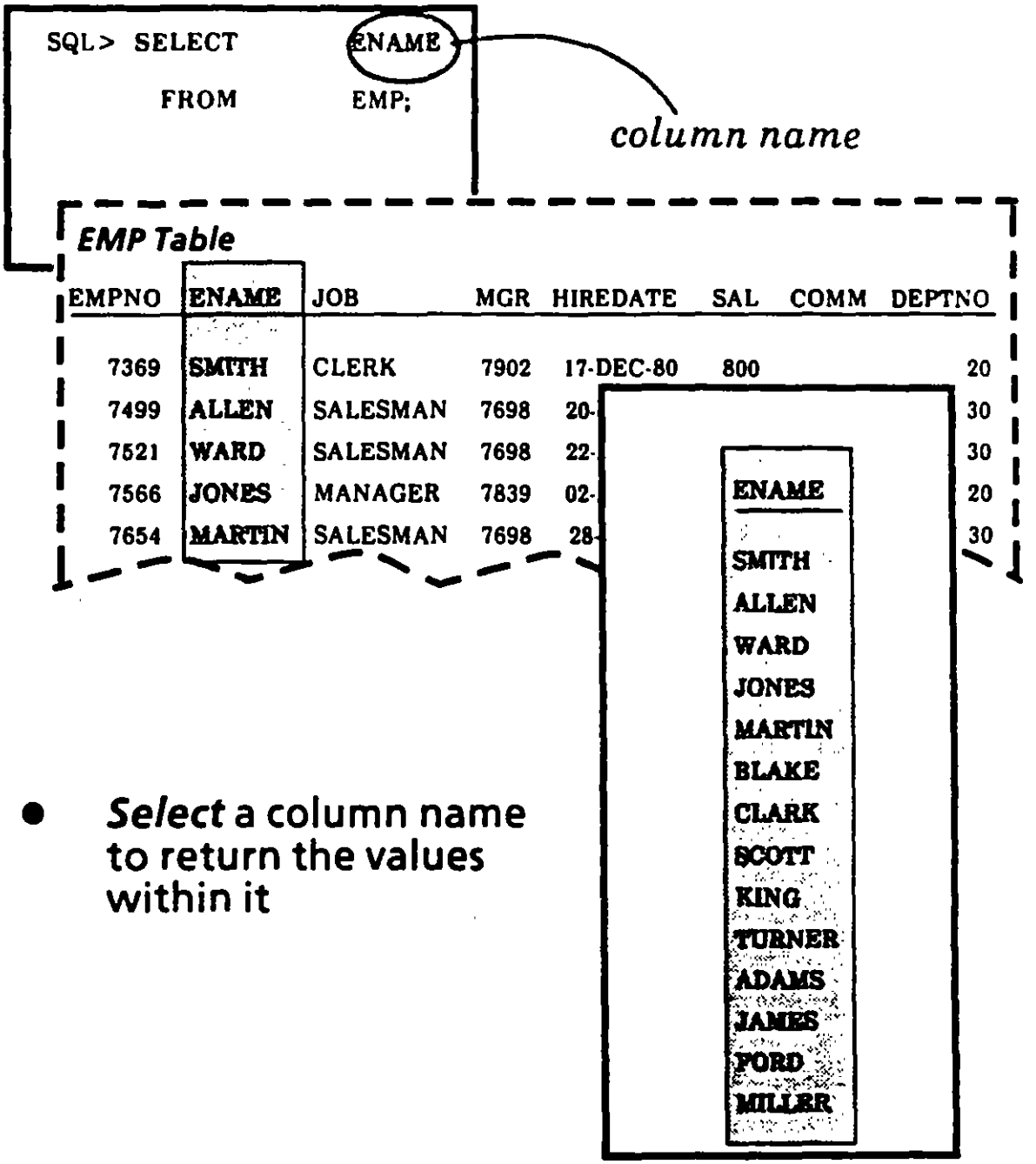
asterisk means *all columns*

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1,400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2,450		10
7788	SCOTT	ANALYST	7566	09-NOV-81	3,000		20
7839	KING	PRESIDENT		17-NOV-81	5,000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1,500	0	30
7876	ADAMS	CLERK	7788	23-SEP-81	1,100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3,000		20
7934	MILLER	CLERK	7782	23-JAN-82	1,300		10

- You use the * to see all columns of the table

Selecting a Specific Column



- *Select* a column name to return the values within it

Selecting Multiple Columns

```
SQL> SELECT    EMPNO, ENAME, JOB
     FROM      EMP;
```

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698				30
7566	JONES	MANAGER	7839				20
7654	MARTIN	SALESMAN	7698				30

EMP Table

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER
7654	MARTIN	SALESMAN
7698	BLAKE	MANAGER
7782	CLARK	MANAGER
7788	SCOTT	ANALYST
7839	KING	PRESIDENT
7844	TURNER	SALESMAN
7876	ADAMS	CLERK
7900	JAMES	CLERK
7902	FORD	ANALYST
7934	MILLER	CLERK

- Place a comma between each column name
- All rows appear for each column selected

Reordering Columns During Selection

```
SQL>  SELECT  EMPNO, ENAME, JOB
      FROM    EMP;
```

EMP Table

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER

```
SQL>  SELECT  JOB, ENAME, EMPNO
      FROM    EMP;
```

EMP Table

JOB	ENAME	EMPNO
CLERK	SMITH	7369
SALESMAN	ALLEN	7499
SALESMAN	WARD	7521
MANAGER	JONES	7566

- The order of selection determines the order of display

Selecting Rows

- If you *don't* specify a ***WHERE*** clause, *all* rows will be selected
- By specifying a ***WHERE*** clause, you can choose specific rows

```
SQL> SELECT    ENAME
     FROM      EMP
     WHERE     DEPTNO = 10;
```

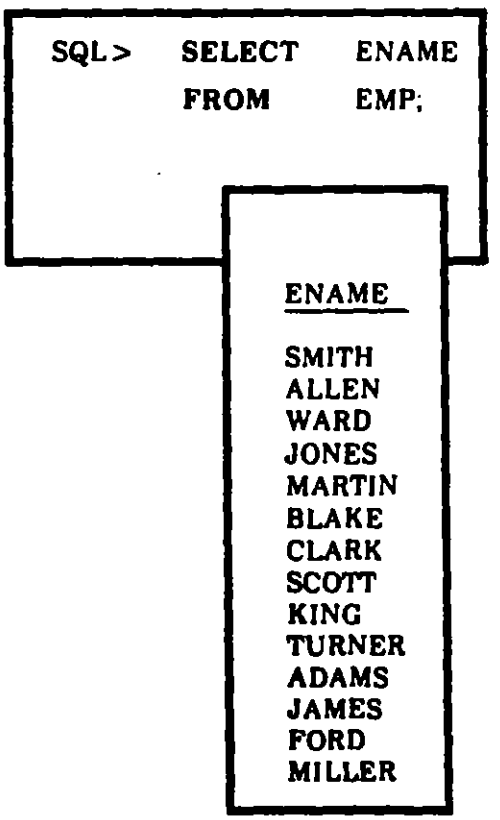
EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2,450		10
7788	SCOTT	ANALYST	7566	09-NOV-81	3,000		20
7839	KING	PRESIDENT		17-NOV-81	5,000		10
7844	TURNER	SALESMAN			,500	0	30
7876	ADAMS	CLERK			,100		20
7900	JAMES	CLERK			950		30
7902	FORD	ANALYST			,000		20
7934	MILLER	CLERK			,300		10

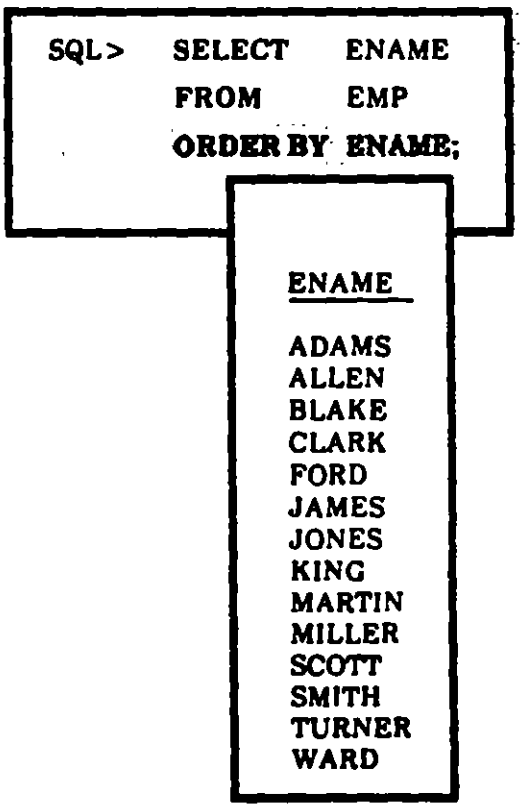
ENAME
CLARK
KING
MILLER

Ordering Rows in a Sequence

- In the relational model, rows have *no particular order*
- The ***ORDER BY*** command is the only way you can ensure rows will be displayed according to specific criteria



Unordered



Ordered

Ordering Rows by Multiple Criteria

- When ordering by multiple criteria,
  - ***Primary Order*** is the first column listed
  - ***Secondary Order*** is the second column listed
  - And so on...
- The default is Ascending Sequence (***ASC***) (A to Z, not Z to A)
- To order in a Descending Sequence, you add the word ***DESC*** after you specify which column to order by

```
SQL> SELECT    *
     FROM      DEPT
     ORDER BY  DEPTNO  DESC;
```

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON
30	SALES	CHICAGO
20	RESEARCH	DALLAS
10	ACCOUNTING	NEW YORK

Selecting Specific Rows: A Single Condition

In the ***WHERE*** clause you can compare a column value to

- a ***CHARACTER*** constant by using single quotes

  where ename = 'SMITH'

  Note: The case you use matters within quoted constants

- an ***ARITHMETIC EXPRESSION*** with no quotes

  where deptno = 20

- another ***COLUMN*** value

  where emp.deptno = dept.deptno
  *(This construct, known as a **JOIN**, will be discussed later.)*

Logical Operators for Selecting Rows

Equality and Inequality Operators

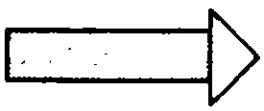
Equal to	=
Not equal to	!= or <>
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=

Other Operators

Equal to any member of the following list	*IN (list)*
Greater than or equal to one value, and less than or equal to another	*BETWEEN low AND high*
Matches the following pattern a string of zero or more characters a string of one character	*LIKE* *%* *_*
Not applicable/missing value	*IS NULL*
Reverses some of the above operators (e.g. *NOT IN, IS NOT NULL*)	*NOT*

Lists of Values: IN and NOT IN

- The ***IN*** operator lets you select rows that match one of the values in a list (uses an ***OR*** evaluation)



Which employees are either clerks or analysts?

```
SQL> SELECT   ENAME, JOB
     FROM     EMP
     WHERE    JOB IN ('CLERK', 'ANALYST');
```

ENAME	JOB
SMITH	CLERK
SCOTT	ANALYST
ADAMS	CLERK
JAMES	CLERK
FORD	ANALYST
MILLER	CLERK

- Conversely, ***NOT IN*** lets you select rows that do not fall in the list

```
SQL> SELECT   ENAME, JOB
     FROM     EMP
     WHERE    JOB NOT IN ('CLERK', 'ANALYST');
```

A Range of Values: BETWEEN and NOT BETWEEN

- The **BETWEEN** operator lets you select rows that contain values within a range.

Which employees earn between $2,000 and $3,000?

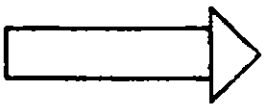
```
SQL>  SELECT  ENAME, JOB,SAL
      FROM    EMP
      WHERE   SAL BETWEEN 2000 AND 3000;
```

- Conversely, **NOT BETWEEN** selects rows outside of a range.

```
SQL>  SELECT  ENAME, JOB,SAL
      FROM    EMP
      WHERE   SAL NOT BETWEEN 2000 AND 3000;
```

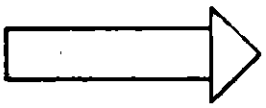
Looking for a Pattern: LIKE and NOT LIKE

- To search for a string of characters , use the **LIKE** operator in the **WHERE** clause



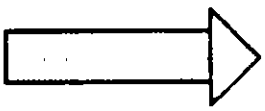
List all employees whose names begin with the letter S.

```
SQL>  SELECT  ENAME, DEPTNO
      FROM    EMP
      WHERE   ENAME LIKE 'S%';
```



List all employees whose names end with the letter K.

```
SQL>  SELECT  ENAME, DEPTNO
      FROM    EMP
      WHERE   ENAME LIKE '%K';
```

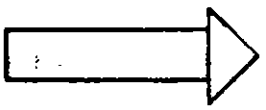


List all employees whose names begin with a W followed by exactly 3 characters.

```
SQL>  SELECT  ENAME, DEPTNO
      FROM    EMP
      WHERE   ENAME LIKE 'W___';
```

- Notice that the wildcard symbol % will match a string of any length, while the _ matches a single character.

- Use **NOT LIKE** to select rows that do not match a pattern.



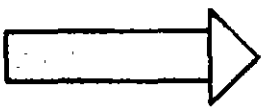
List all employees whose job titles do not begin with the string SALES.

```
SQL>   SELECT  ENAME, JOB
       FROM    EMP
       WHERE   JOB NOT LIKE 'SALES%';
```

Write a query below to list all employees whose names begin with the letter M, end with the letter R, and have an L as the third letter.

NULL Values: IS NULL and IS NOT NULL

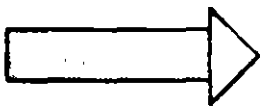
- A ***NULL value*** in a number column is not the same as a zero.
- Zero is a number; ***NULL*** is not a number
- ***NULL*** means that the value is unknown, missing, or not applicable; it should not be treated like a zero.
- Searching for ***NULL values***:



List all employees who are not eligible for commission (ie., whose commission column is NULL).

```
SQL>  SELECT  ENAME, JOB
      FROM    EMP
      WHERE   COMM IS NULL;
```

- Searching for ***non-NULL values***:



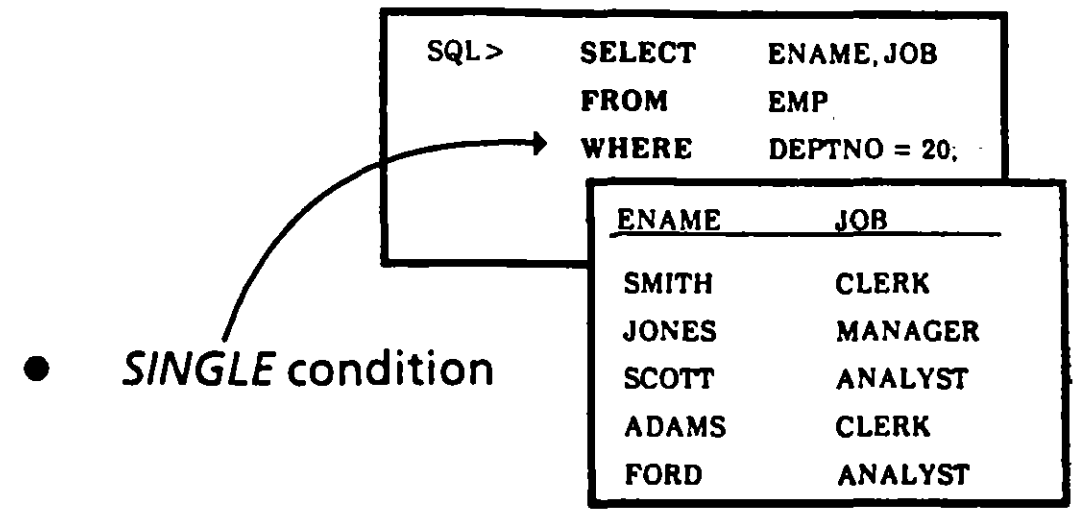
List all employees who are eligible for commission.

```
SQL>  SELECT  ENAME, JOB
      FROM    EMP
      WHERE   COMM IS NOT NULL;
```

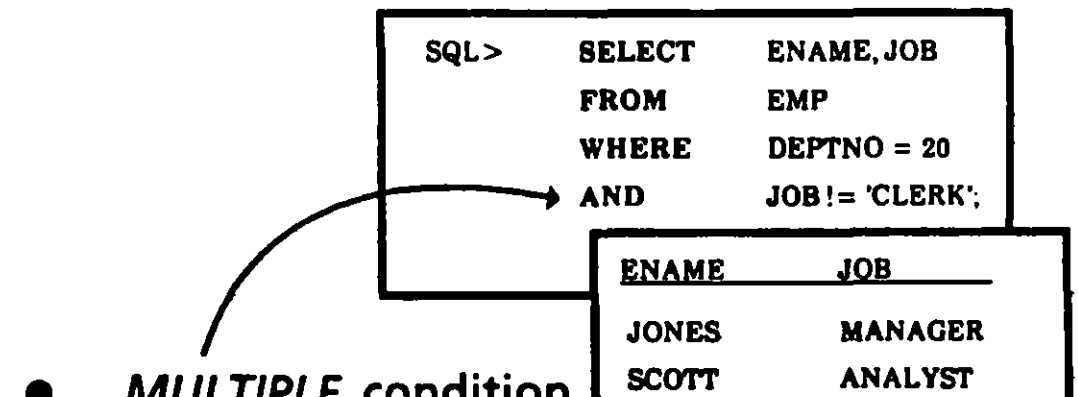
- Note: ***NULL*** values do not take up any storage space in an Oracle RDBMS.

Multiple Conditions

- The ***WHERE*** clause can further qualify what rows are returned by specifying more than one condition needed to satisfy the query



- *SINGLE* condition



- *MULTIPLE* condition

- The *AND* operator adds further criteria selected rows must meet.

- In contrast, the *OR* operator specifies selection of rows where either condition is met.

```
SELECT    ENAME,JOB
FROM      EMP
WHERE     DEPTNO = 20
AND       JOB != 'CLERK';
```

ENAME	JOB
JONES	MANAGER
SCOTT	ANALYST
FORD	ANALYST

```
SELECT    ENAME,JOB
FROM      EMP
WHERE     DEPTNO = 20
OR        JOB != 'CLERK';
```

ENAME	JOB
SMITH	CLERK
ALLEN	SALESMAN
WARD	SALESMAN
JONES	MANAGER
MARTIN	SALESMAN
BLAKE	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
TURNER	SALESMAN
ADAMS	CLERK
FORD	ANALYST

Precedence of Multiple Operators

- You may use ***AND*** and ***OR*** in a single ***WHERE*** clause

- You should use ( ) to establish precedence

- Order of evaluation if you *don't* specify ( ):

  First ***AND***

  Then ***OR***

- For Example:

```
WHERE   DEPTNO   =   30
AND     JOB      =   'SALESMAN'
OR      SAL      >   2000
```

This refers to *salesmen in dept 30 or any employee who makes over $2,000*

It doesn't refer to *anyone in dept 30 who is either a salesman or who makes over $2,000*

Introduction to Expressions

- You can use SQL like a calculator to express values with arithmetic operators. These include:

+	*ADD*
-	*SUBTRACT*
*	*MULTIPLY*
/	*DIVIDE*

- These operators can be used in just about any kind of clause, including

  SELECT

  WHERE

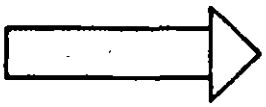
  ORDER BY

  and other clauses you will learn about later.

  Note: It does not make sense to use an arithmetic operator in the ***FROM*** clause.

Numeric Expressions

- You can use more than one expression in the same query:



Who earns a commission that is more than 5% of their base salary?

```
SQL> SELECT    ENAME, SAL, COMM,
               COMM/SAL
     FROM      EMP
     WHERE     COMM > .05*SAL
     ORDER BY  COMM/SAL DESC;
```

ENAME	SAL	COMM	COMM/SAL
MARTIN	1,250	1,400	1.12
WARD	1,250	500	.4
ALLEN	1,600	300	.1875

Numeric Expressions with Multiple Operators

- You may nest multiple expressions together

- Here's the ***Order of Evaluation:***

  Evaluated First:

Multiplication	*
Division	/

  Evaluated Next:

Addition	+
Subtraction	-

- Evaluation occurs from *left to right*, wherever the above ordering of evaluation does not apply

- You can control the order of evaluation with the use of ***( )***

  For instance:

  12*(SAL + COMM) != 12*SAL + COMM

Using Expressions in DATE Arithmetic

- You can use expressions to specify values that involve ***DATE*** values -- arithmetic uses days as the basic unit

- You can both add and subtract dates

- Here are some examples using addition:

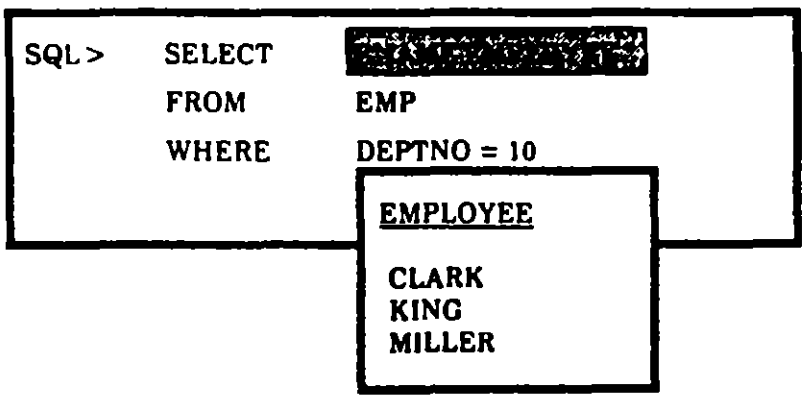
Add two days to 6-Mar-87
6-MAR-87 + 2 = 8-MAR-87

Add two hours to 6-Mar-87
6-MAR-87 + 2/24 = 6-MAR-87 and 2 hrs

Add15 seconds to 6-Mar-87
6-MAR-87
+ 15/(24*60*60) = 6-MAR-87 and 15 secs

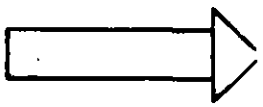
Using Column Aliases

- The column heading that displays normally reflects the one that you specified when you created the table
- You can make a different column heading display by specifying a ***COLUMN ALIAS*** in the ***SELECT*** clause
- To specify an alias, follow the column name with a blank and then the alias name you want
- Here's an example:



Column Aliases and Expressions

- If a ***SELECT*** clause contains an arithmetic expression, that expression is used as the display heading. An alias can be used to temporarily "rename" a computed column to make query results more readable.



Who earns a commission that is more than 5% of their base salary?

```
SQL> SELECT    ENAME, SAL, COMM, COMM/SAL "C/S RATIO"
     FROM      EMP
     WHERE     COMM > .05*SAL
     ORDER BY  COMM/SAL DESC;
```

ENAME	SAL	COMM	C/S RATIO
MARTIN	1,250	1,400	1.12
WARD	1,250	500	.4
ALLEN	1,600	300	.1875

- Notice that the expression itself, not the alias, is referenced in the ***ORDER BY*** clause. This rule also applies to all other clauses in the ***SELECT*** statement.

- Also notice that an alias containing special characters such as a blank and a slash (/) must be enclosed in double quotes.

- The alias affects the ***SELECT*** command in which it is used. It has no effect on other queries.

General Query Syntax: A Summary

- The overall syntax is very simple:

SELECT ...
FROM ...
[*WHERE ...*]
[*GROUP BY ...*
[*HAVING ...*]]
[*ORDER BY ...*]

- The formal syntax that you have learned so far looks like this:

SELECT	**col 1,col 2...col n**
FROM	**tablename**
WHERE	**conditions are true**
ORDER BY	**col 1, col 2...col n**

Background for Editing SQL Statements

- ***SQL BUFFER***
  This buffer holds the ***CURRENT SQL COMMAND*** until you enter another SQL command or exit

- ***SQL*Plus*** commands for editing

  LIST or ***L*** - Display the contents of the *SQL Buffer*

  LIST 4 - List the fourth line of the Current SQL Command

  CHANGE or ***C*** - Change first occurrence of text in a single line. Note: Using ... will match a string of any length. e.g. SQL> C/(...)/('ANALYST')/

  INPUT or ***I*** - Add one or more lines to the Current SQL Command

  APPEND or ***A*** - Add text to a line

  DEL - Delete one line

  RUN - List and run the statement in the *SQL Buffer*

  / - Run the statement in the *SQL Buffer*

  EDIT - Write the statement in the *SQL BUFFER* to an operating system text file and call the standard operating system editor to allow changes

  You are ready to complete **Practice Session One.**

3. Data Manipulation

Inserting Data

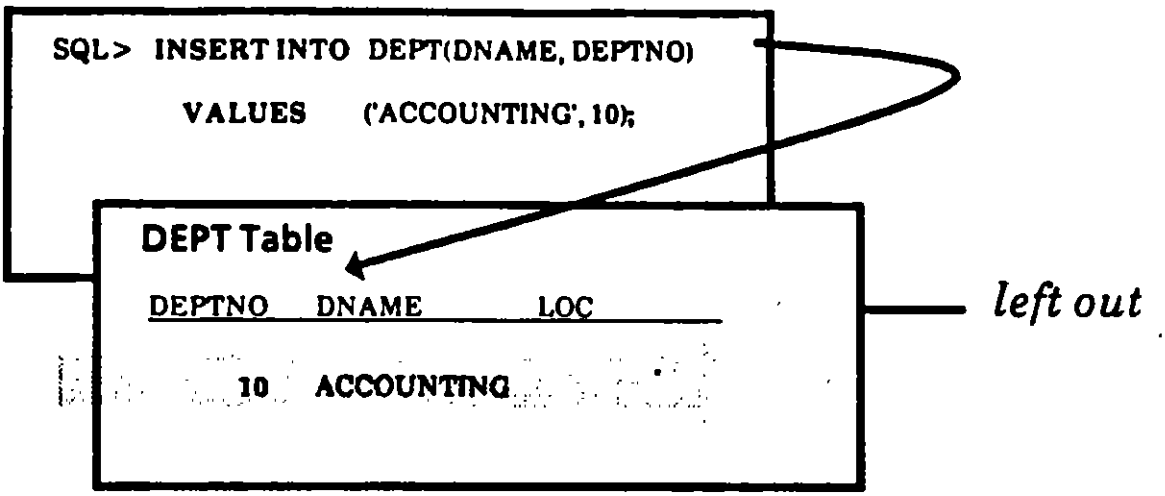
```
SQL> INSERT INTO DEPT
       VALUES    (10,'ACCOUNTING',
                  'NEW YORK');
```

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK

- A table must be created before you can insert data
- Separate values with commas
- Use the ***DESCRIBE*** command to show order and type of columns
- Each value must match the data type of targeted column
- Enclose **CHAR** and **DATE** values in single quotes

More about Inserting Data



- List column names in ***INSERT*** clause to
  - forego data entry for one or more columns
  - enter data in a specific sequence

- Values must be listed in the order specified by the ***INSERT*** clause

Inserting a Row from Another Table

- You can use the ***INSERT*** command with a query to select rows from one table and insert them into another table.

- The query replaces the ***VALUES*** clause.

```
SQL> INSERT INTO EMP (EMPNO, ENAME, DEPTNO)
     SELECT ID, NAME, DEPARTMENT
     FROM OLD_EMP
     WHERE DEPARTMENT IN (10, 20, 30, 40);
```

Using Parameters in the INSERT Command

- An ***INSERT*** command may contain parameters representing values to be provided when the command is run.

- Each parameter consists of an & which is usually followed by the name of the column.

```
SQL>  INSERT INTO DEPT
      VALUES (&DEPTNO, &DNAME, &LOC);
```

Execution of this statement causes ***SQL*Plus*** to prompt you for values for each of the parameters.

- Repeated execution allows the user to insert multiple rows into a table quickly.

- You do not have to put quotes around a **CHAR** or **DATE** value if you put them around the parameter.

```
SQL>  INSERT INTO DEPT
      VALUES (&DEPTNO, '&DNAME', '&LOC');
```

Inserting NULL Values

- If you don't include a column in the ***INSERT*** clause, the value for that column defaults to ***NULL***

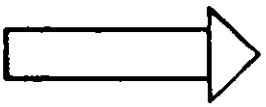
- You may list ***NULL*** in the ***VALUES*** clause (unless you specified ***NOT NULL*** for that column)

```
SQL>  INSERT INTO DEPT
      VALUES (50, 'EDUCATION', NULL);
```

Inserting DATE Values

- Default format for entering dates:

'DD-MON-YY'

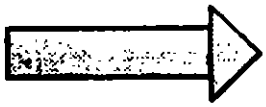


Enter a new employee 'STONE'.

```
SQL> INSERT INTO EMP
     (EMPNO, ENAME, HIREDATE)
     VALUES (7963, 'STONE', '07-APR-87');
```

- To automatically enter today's date and time:

SYSDATE



Enter a new employee 'KOHN', who was hired today.

```
SQL> INSERT INTO EMP
     (EMPNO, ENAME, HIREDATE)
     VALUES (7600, 'KOHN', SYSDATE);
```

Updating Fields

Promote Martin to Manager

```
SQL>  UPDATE   EMP
      SET      JOB     = 'MANAGER'
      WHERE    ENAME   = 'MARTIN';
```

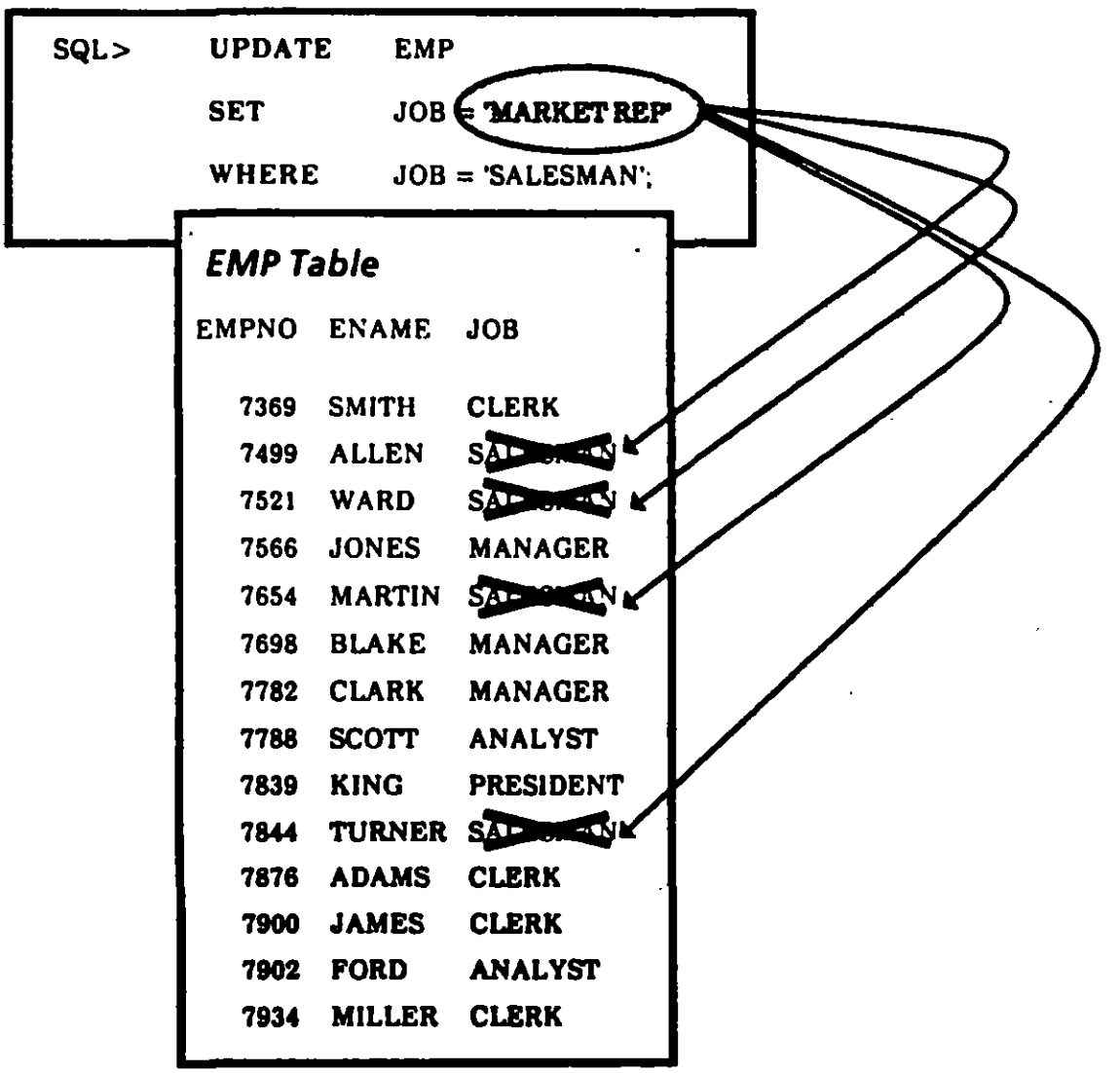
EMP Table

EMPNO	ENAME	JOB
7369	SMITH	CLERK
7499	ALLEN	SALESMAN
7521	WARD	SALESMAN
7566	JONES	MANAGER
7654	MARTIN	SALESMAN
7698	BLAKE	MANAGER
7782	CLARK	MANAGER
7788	SCOTT	ANALYST
7839	KING	PRESIDENT
7844	TURNER	SALESMAN
7876	ADAMS	CLERK
7900	JAMES	CLERK
7902	FORD	ANALYST
7934	MILLER	CLERK

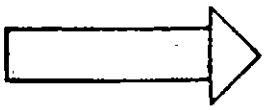
- If you omit the ***WHERE*** clause all values in the column will change to the value in the ***SET*** clause

Updating Multiple Rows

Change all Salesmen to Market Reps.



Updating Multiple Columns



Change all Salesmen to Market Reps, and transfer them to Dept 40.

```
SQL>    UPDATE    EMP
        SET       DEPTNO = 40, JOB = 'MARKET REP'
        WHERE     JOB = 'SALESMAN';
```

EMP Table

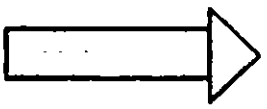
DEPTNO	ENAME	JOB
20	SMITH	CLERK
30	ALLEN	SALESMAN
30	WARD	SALESMAN
20	JONES	MANAGER
30	MARTIN	SALESMAN
30	BLAKE	MANAGER
10	CLARK	MANAGER
20	SCOTT	ANALYST
10	KING	PRESIDENT
30	TURNER	SALESMAN
20	ADAMS	CLERK
30	JAMES	CLERK
20	FORD	ANALYST
10	MILLER	CLERK

- You can update multiple columns by specifying your changes in the *SET* clause

Deleting Rows

- You cannot delete partial rows. Instead, update the column to NULL.
- The ***WHERE*** clause determines which rows are deleted:

  If you forget the ***WHERE*** clause, you will delete all the rows



Martin quit. Remove him from the company roster.

```
SQL>    DELETE FROM    EMP
        WHERE          EMPNO = 7654;
```

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE
7369	SMITH	CLERK	7902	17-DEC-80
7499	ALLEN	SALESMAN	7698	20-FEB-81
7521	WARD	SALESMAN	7698	22-FEB-81
7566	JONES	MANAGER	7839	02-APR-81
7654	MARTIN	SALESMAN	7698	28-SEP-81
7698	BLAKE	MANAGER	7839	01-MAY-81
7782	CLARK	MANAGER	7839	09-JUN-81

Controlling When Changes Take Effect: COMMIT

- Insertions, deletions and updates to tables are not made permanent until work is committed to the database.

- Until work is committed, only the user who has made changes to tables can see those changes; other users see the data as it was as of the last COMMIT.

- A ***COMMIT*** may be ***explicit, implicit*** or ***automatic***:

  Explicit COMMIT -

  Issue the ***SQL*** command ***COMMIT*** to make all pending changes permanent.

```
SQL> COMMIT
```

Implicit COMMIT -

The following *SQL* commands cause an implicit commit:

> *ALTER, AUDIT, COMMENT, CONNECT, CREATE, DISCONNECT, DROP, EXIT, GRANT, NOAUDIT, QUIT, REVOKE, and RENAME.*

Automatic *COMMIT* -

Changes will take effect immediately after an *INSERT, UPDATE*, or *DELETE* is executed if *AUTOCOMMIT* is enabled. Use the *SQL*Plus SET* command:

```
SQL> SET AUTOCOMMIT ON
```

Note: *SET* commands are covered in Chapter 5.

Controlling When Changes Take Effect: ROLLBACK

- The *SQL* command ***ROLLBACK*** cancels all pending changes by "rolling back" work.

- On ***ROLLBACK*** the database is restored to its state at the time of the last ***COMMIT***.

```
SQL> ROLLBACK
```

Logical Transactions

- All changes to the database between successive ***COMMIT*** operations are called a ***transaction***.

- When a transaction is interrupted by a serious error, such as a system failure, the entire transaction is automatically rolled back.

- This prevents the error from causing only part of a ***logical transaction*** to be committed.

- Examples of a ***logical transaction***:
  - Withdrawal of funds from a savings account followed by deposit of those funds into a checking account.
  - Creating a new department in a company: Transaction consists of updating the deptno column in the EMP table and inserting a new row in the DEPT table.

You are ready to complete Practice Session Two.

4. Creating Tables and Views

How to Create a Table

```
SQL> CREATE TABLE DEPT
        (DEPTNO    NUMBER (2),
        DNAME      CHAR (14),
        LOC        CHAR (13));
```

DEPT Table

DEPTNO	DNAME	LOC

- When you create a table you specify the
  - table name
  - column names
  - type of values in the column
  - maximum width of the column
- The ***DATA DICTIONARY*** is automatically updated
- A table may have up to 254 columns

Rules for Naming Tables and Columns

- ***Restrictions on names***

  First character must be ***A-Z*** or ***a-z*** (but all are stored in uppercase)

  Subsequent characters can also be ***numbers*** or ***$***, ***#***, and ***_*** (commas not allowed)

  Names can be up to 30 characters

- ***Names must be unique***

  Your table name must be unique to your userid

  It may not duplicate Oracle reserved words

  Column names must be unique within a table

- ***Double quotes can be used***

  If you enclose the table name in double quotes, none of the above character rules apply

  The case you use only makes a difference when you use double quotes

  All subsequent access to objects named in this way will require the use of double quotes as well

Common Data Types

- ***CHAR*** *(n)*
  n = maximum length of char strings
  You can make *n* any length up to 240 characters
  Don't limit *n* - storage isn't affected

- ***NUMBER*** *(n,d)*
  n = maximum number of digits
  d = maximum number of digits right of decimal

- ***DATE*** consists of
  The relevant date plus the actual time of day

- ***LONG***
  Up to 65,536 characters (64K) may be stored per field

- ***RAW***
  Raw binary data

Forbidding NULL Values

- Sometimes you want to make sure a column does not end up with any NULL values

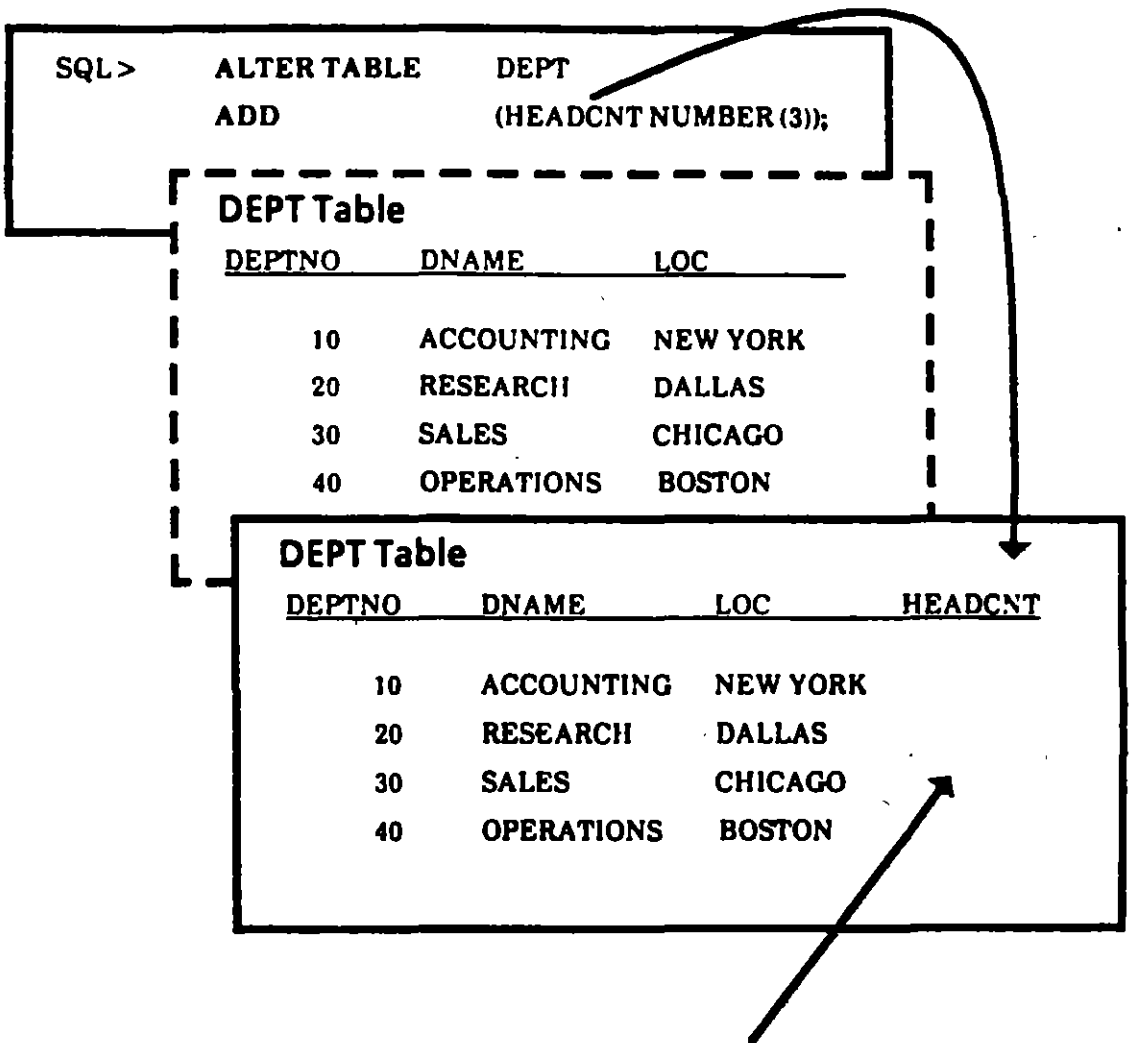
  For instance, you might want to forbid a row from being inserted into the *EMP* table if the *Employee Number* is unknown

- To forbid NULL values in a column, enter the ***NOT NULL*** clause at the end of the column information

  Then any attempt to add a row without a value for EMPNO to the EMP table will fail and produce an error message

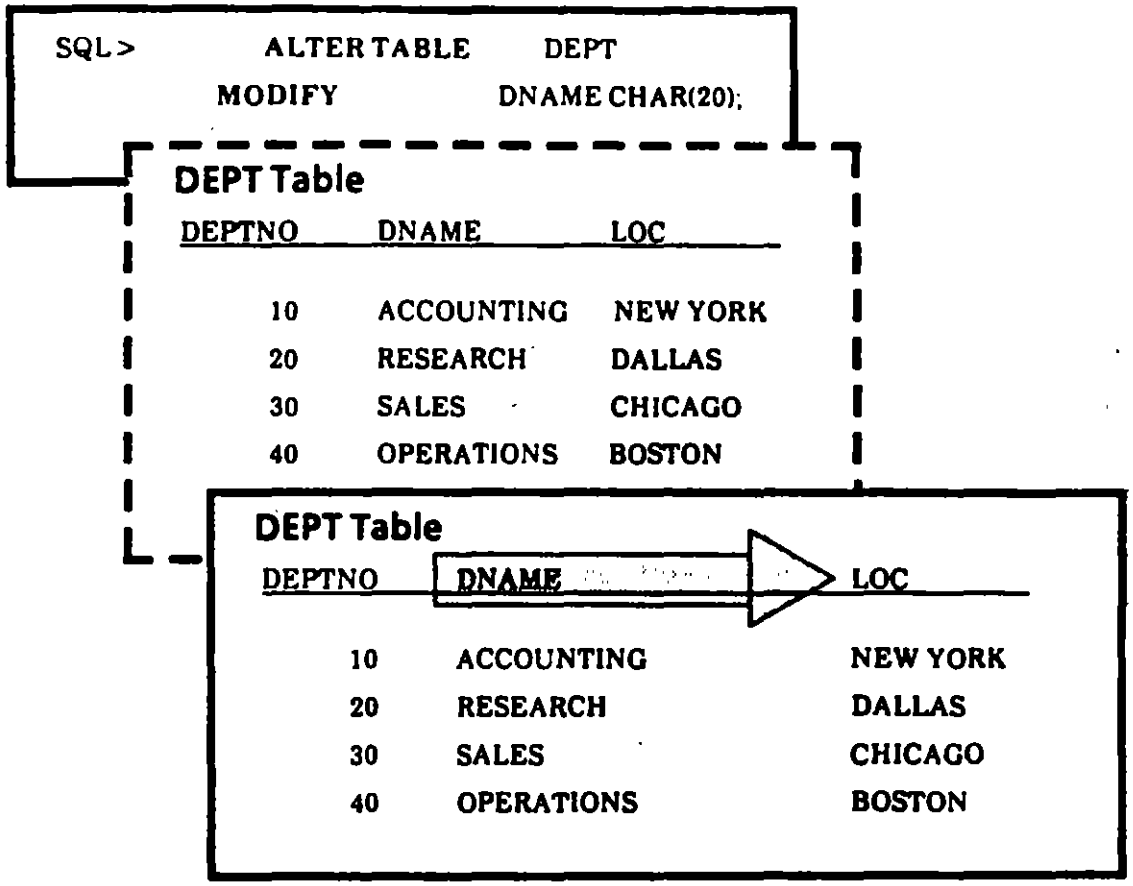
```
SQL>    CREATE TABLE    DEPT
        (DEPTNO NUMBER (2) NOT NULL,
        DNAME   CHAR(14),
        LOC     CHAR(13));
```

Adding a New Column To a Table



- When a new column is first added, all of its rows are ***NULL***

Enlarging a Column: ALTER TABLE and MODIFY



- ***ALTER*** the ***TABLE*** to change the column size
- You may not reduce unless column is empty
- Nor may you change datatype unless the column is empty
- You cannot modify a column to make it ***NOT NULL*** unless all rows have values in that column

Views

- A *View* is like a window through which you can view or change information in a table

- A *View* is a *Virtual Table*
  - It *looks like* a table, but it does not exist as such
  - Its data are *derived* from tables
  - However, there is *no copy* of the data

- A *View* provides
  - Simplicity - to see exactly what you need
  - Security - to prevent unauthorized users from seeing certain columns or rows

One Table, Multiple Views

- You can have multiple views of the same table

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	[illegible]	[illegible]	7839	02-APR-81	[illegible]		[illegible]
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1,400	30
7698	[illegible]	[illegible]	7839	01-MAY-81	[illegible]		30
7782	[illegible]	[illegible]	7839	09-JUN-81	[illegible]		[illegible]
7788	SCOTT	ANALYST	7566	09-NOV-81	3,000		[illegible]
7839	KING	PRESIDENT		17-NOV-81	5,000		[illegible]
7844	TURNER	SALESMAN	7698	08-SEP-81	1,500	0	[illegible]
7876	ADAMS	CLERK	7788	23-SEP-81	1,100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3,000		20
7934	MILLER	CLERK	7782	23-JAN-82	1,300		10

ENAME	JOB
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK

View of Clerks

[illegible]

View of Managers

Creating, Naming, and Looking at Views

- Creating a ***VIEW*** is different than creating a table:

```
SQL> CREATE VIEW MANAGERS AS
     SELECT      ENAME, JOB, SAL
     FROM        EMP
     WHERE       JOB = 'MANAGER';

VIEW CREATED
```

- You may use any valid ***SELECT*** statement which does not contain an ***ORDER BY*** clause when creating a *View*

- To look at a view you ***SELECT*** information as if the ***VIEW*** were a***TABLE***

```
SQL> SELECT *
     FROM   MANAGERS;

ENAME      JOB        SAL
JONES      MANAGER    2,975
BLAKE      MANAGER    2,850
CLARK      MANAGER    2,450
```

Creating Views with Alias Column Names

- Unless otherwise specified, a view inherits the column names from the table on which it is based.

- To give the view column names different than those in the base table, use the following syntax:

CREATE VIEW viewname (alias, alias,...)
AS query;

```
SQL> CREATE VIEW MYDEPT
     (PERSON, TITLE, SALARY)
     AS SELECT ENAME, JOB, SAL
     FROM EMP
     WHERE DEPTNO = 10;
```

ACTUALIZANDO UNA VISTA

Es posible borrar renglones de una tabla a través de una vista, si la consulta (QUERY) usada para definir la vista cumple que:

- Selecciona renglones de solo una tabla.
- No contiene una clausula GROUP BY o DISTINCT, una función de grupo o una referencia a la Pseudocolumna ROWNUM.

Se pueden actualizar renglones de una tabla a través de una vista, si la consulta usada para definirla observa las dos restricciones anteriores, mas la siguiente:

- No define alguna de las columnas que son actualizadas, con una expresión.

Se puede insertar renglones en una tabla, a través de una vista, si la consulta usada cumple las tres restricciones anteriores, mas:

- Cualquier columna NO NULA (not null) definida en la tabla es representada en la vista.

More About Views: WITH CHECK OPTION

- The ***WITH CHECK OPTION*** specifies that inserts and updates performed through the view should not be allowed to manipulate data that the view cannot select.

```
SQL> CREATE VIEW DEPT20 AS
     SELECT ENAME, JOB, SAL, DEPTNO
     FROM EMP
     WHERE DEPTNO = 20
     WITH CHECK OPTION;
```

Now, the following update statement would produce an error.

```
SQL> UPDATE DEPT20
     SET DEPTNO = 30
     WHERE ENAME = 'WARD';
```

Copying Tables & Views

- Reasons for Copying Tables and Views

  Back up the original

  Give a copy to someone else

  Make tentative changes

  Archive them before removal

- To copy a Table

  Use the ***CREATE TABLE*** command with an ***AS*** clause followed by a subquery

  In Chapter 7 you will learn about subqueries

  When the table is copied, it includes the column definitions and the data

```
SQL> CREATE TABLE      EMP2
     AS SELECT * FROM  EMP;
```

Dropping Tables and Views

- **Dropping *TABLES***

  The SQL command is ***DROP TABLE*** *table name*

  If you have data in the table, it will be deleted permanently

  Once the table is gone, you cannot get it back

- **Dropping *VIEWS***

  The SQL command is ***DROP VIEW*** *view name*

  The *TABLE(s)* on which the *VIEW* is based will not be changed

  You are ready to complete **Practice Session Three.**

5. SQL*Plus Reporting

The Limitations of SQL for Report Formatting

Show the employees in each department who have salaries exceeding $2,000.

```
SQL> SELECT   DEPTNO, ENAME, SAL
     FROM     EMP
     WHERE    SAL > 2000;
```

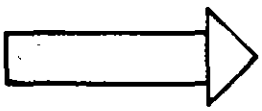
EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1,600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1,250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2,975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1,250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2,850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2,450		10
7788	SCOTT	ANALYST	7				20
7839	KING	PRESIDENT					10
7844	TURNER	SALESMAN	7				30
7876	ADAMS	CLERK	7				20
7900	JAMES	CLERK	7				30
7902	FORD	ANALYST	7				20
7934	MILLER	CLERK	7				10

DEPTNO	ENAME	SAL
10	KING	5000
10	CLARK	2450
20	FORD	3000
20	SCOTT	3000
20	JONES	2975
30	BLAKE	2850

- ***SQL*** **only provides raw information**
- **In contrast, *SQL*Plus* allows you to customize your reports**

*Report Formatting with SQL*Plus*



Show the employees in each department who have salaries exceeding $2,000.

MON MAR 12 page 1

SAMPLE REPORT for
HITECH CORP.

DEPARTMENT	NAME	SALARY
10	KING	$5,000
	CLARK	2,450
**********		
sum		$7,450
20	FORD	$3,000
	SCOTT	3,000
	JONES	2,975
**********		
sum		$8,975
30	BLAKE	$2,850
**********		
sum		$2,850

STRICTLY CONFIDENTIAL

A Sample of SQL*Plus Format Commands

```
SQL>  COLUMN  DEPTNO  HEADING  DEPARTMENT
SQL>  COLUMN  ENAME   HEADING  NAME
SQL>  COLUMN  SAL     HEADING  SALARY
SQL>  COLUMN  SAL     FORMAT   $99,999.00
SQL>  TTITLE  'SAMPLE REPORT for | HITECH CORP.'
SQL>  BTITLE  'STRICTLY CONFIDENTIAL'
SQL>  BREAK ON DEPTNO
SQL>  COMPUTE SUM OF SAL ON DEPTNO
SQL>  RUN
```

- A semicolon is not required after a *SQL*Plus* command.
- A *SQL* statement must be run to see the effects of *SQL*Plus* format commands.

Headings & Footers

```
SQL> TTITLE          'SAMPLE REPORT for | HITECH CORP'  <return>
SQL> BTITLE RIGHT    'STRICTLY CONFIDENTIAL'            <return>
```

Options:
LEFT
CENTER
RIGHT

Each vertical line produces a line feed

No semicolon needed

- Date and page number automatically appear at top of page when using this form of ***TTITLE***

- Both title commands remain enabled until

  you reset to another title

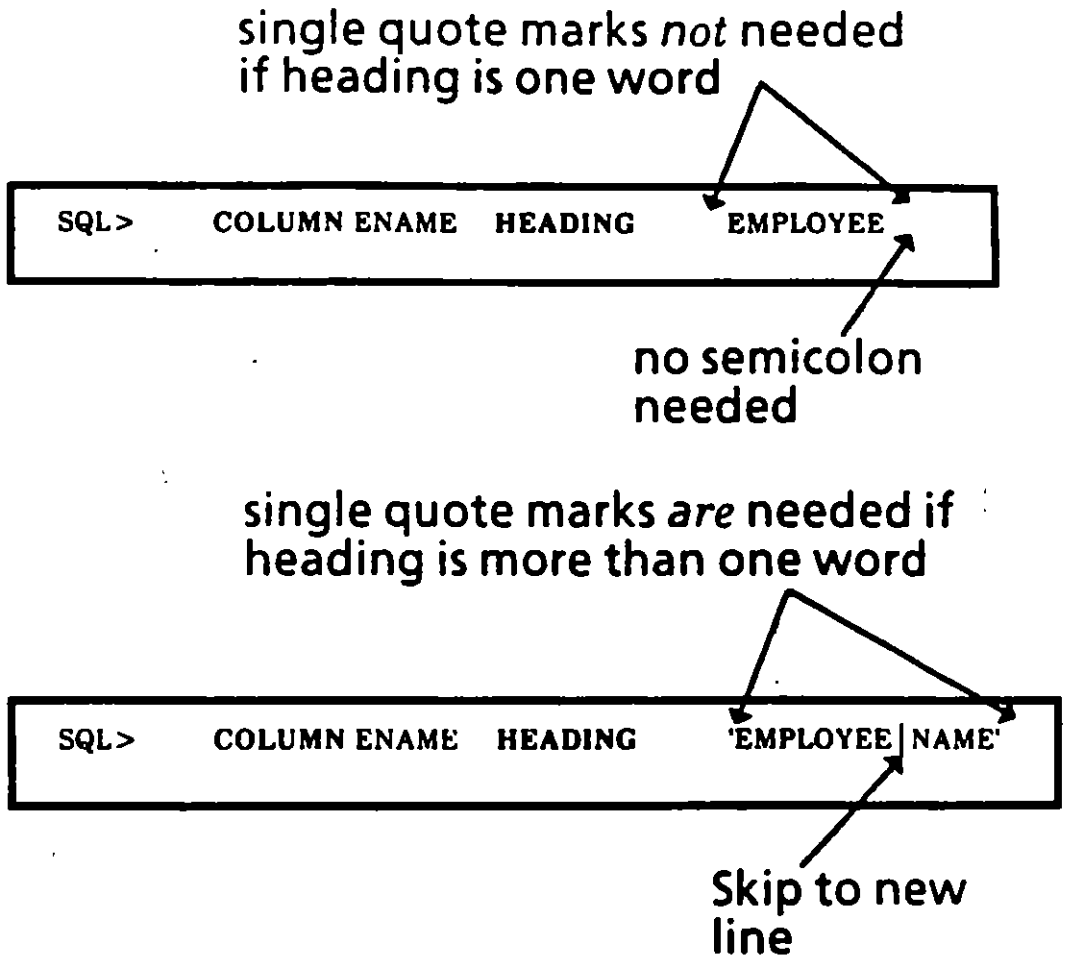
  session is over

  title commands are disabled:

  TTITLE OFF
  BTITLE OFF

Column Names

- Column name will correspond to ***HEADING***

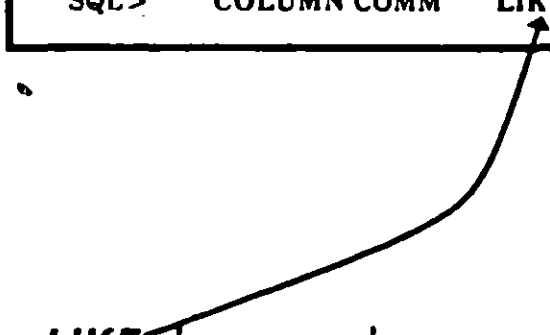


- You can undo a column specification by clearing it

```
SQL> COLUMN ENAME CLEAR
```

Formatting Columns

```
SQL>  COLUMN ENAME  FORMAT  A15
SQL>  COLUMN SAL    FORMAT  $9,999.99
SQL>  COLUMN COMM   LIKE      SAL
```



- The ***LIKE*** clause makes one column look exactly like another
- It copies the format and heading of the column

Ordering Rows Together

```
SQL>    BREAK ON     DEPTNO

SQL>    SELECT       DEPTNO, ENAME
        FROM         EMP
        ORDER BY     DEPTNO;
```

no semicolon

semicolon

EMP Table

DEPTNO	ENAME
10	CLARK
	MILLER
	KING
20	SMITH
	SCOTT
	JONES
	ADAMS
	FORD
30	ALLEN
	BLAKE
	TURNER
	JAMES
	MARTIN
	WARD

- ***BREAK*** can suppress repetition of values
- ***ORDER BY*** clause is needed to control the break
- Only one break command may be in effect, but you may break on more than one column by:

BREAK ON col1 ON col2

Breaking Up Sets of Rows

```
SQL> BREAK ON DEPTNO SKIP 2

SQL> SELECT DEPTNO, ENAME
     FROM EMP
     ORDER BY DEPTNO;
```

EMP Table

DEPTNO	ENAME
10	CLARK
	MILLER
	KING
20	SMITH
	SCOTT
	JONES
	ADAMS
	FORD
30	ALLEN
	BLAKE
	TURNER
	JAMES
	MARTIN
	WARD

- Double spacing between departments
- ***CLEAR BREAK*** erases and disables the ***BREAK*** command
- May break on pages by ***BREAK ON PAGE***
- May break on report by ***BREAK ON REPORT***
- May break on several conditions, for example: ***BREAK ON PAGE ON REPORT***

Displaying Computations on Groups

```
SQL> BREAK ON DEPTNO SKIP 2

SQL> COMPUTE SUM OF SAL ON DEPTNO

SQL> SELECT DEPTNO, ENAME, SAL
     FROM EMP
     ORDER BY DEPTNO;
```

- Sums the *Salaries* for each department
- The *SKIP* clause allows you to see the rows for each department more distinctly

Other COMPUTE Commands

- ***COMPUTE*** ***AVG*** ***OF*** sal ***ON*** deptno

COUNT	number of non-null values
MAX	highest value
MIN	lowest value
STD	standard deviation
VAR	variance
NUMBER	number of rows

- ***CLEAR COMPUTES***
  - ***COMPUTE*** is left enabled until
    - it is disabled
    - it is respecified
    - session ends
  - To disable it, enter ***CLEAR COMPUTE***

*SQL*Plus Environment Commands: SHOW and SET*

- The current *SQL*Plus* session settings can be displayed by entering:

SHOW option

```
SQL> SHOW AUTOCOMMIT
```

- ***SHOW ALL*** - displays all settings

- Specify settings with the ***SET*** command:

SET option value

```
SQL> SET AUTOCOMMIT ON
```

- *SET* commands include (default settings are underlined):

SET AUTOCOMMIT{OFF|ON|IMMEDIATE}

ON or *IMM* - enables automatic committing of changes.

SET ECHOSET {OFF|ON}

ON - commands executed from a command file will be displayed to the terminal

OFF - suppresses display of commands

SET FEEDBACK{OFF|ON}

ON - query results will be followed by message indicating number of records returned

OFF - message will be suppressed

SET HEADING {OFF|ON}

ON - column headings displayed in reports

OFF - headings suppressed

SET LINESIZE {n}

Sets number of characters ***SQL*Plus*** will display on a line - default is 80

SET PAGESIZE {n}

Sets number of lines per page - default is 14

- *SET* commands (continued):

  SET PAUSE {OFF|ON|text}

  ON - forces user to press [Return] before displaying next page of output

  OFF - suppresses wait

  Text - specifies message to be displayed as prompt

  SET BUFFER buffer

  Buffer becomes the current buffer. ***SQL*** is the default buffer.

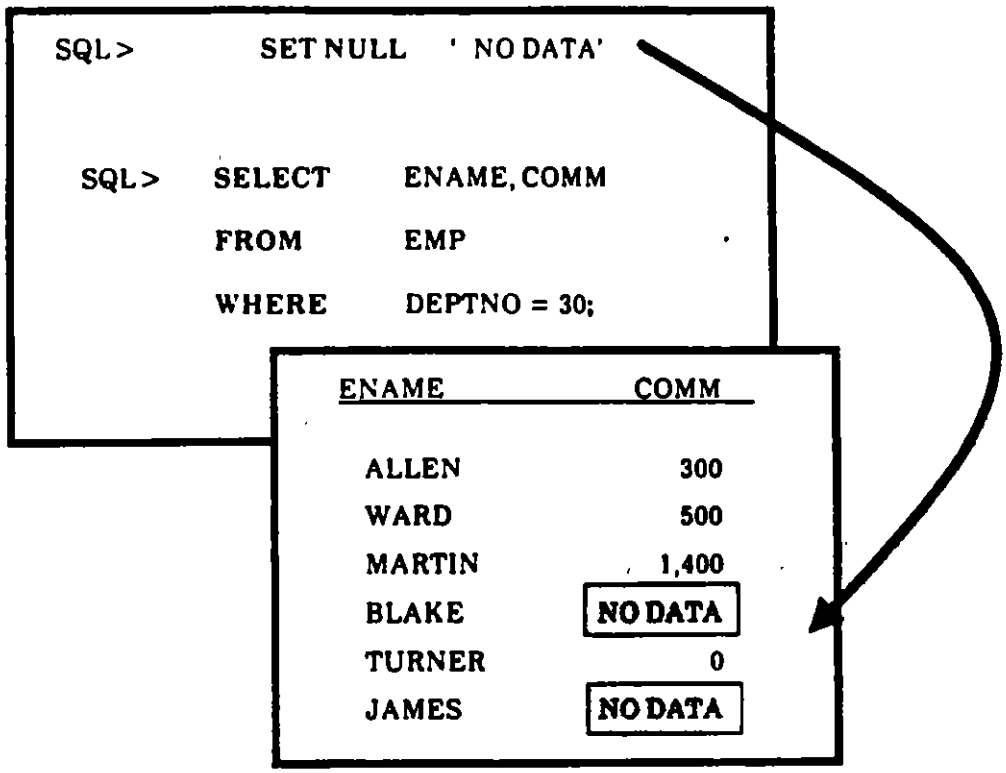
  The ***SQL*** buffer can hold only one ***SQL*** command at a time. Use another buffer if you want to input ***SQL*Plus*** commands in addition to a ***SQL*** command

- Settings remain in effect for the current ***SQL*Plus*** session.

- Frequently used settings may be placed in the ***LOGIN.SQL*** file.

SET NULL

List "NO DATA" for employees in Dept 30 who don't earn a commission



- *SET NULL* is a SQL*Plus command
- You use it to note missing values
- The substitute can be any string

Saving Commands to a File: *The SAVE Command*

- To save a *SQL* command on disk use the *SAVE* command:

  SAVE filename

- The suffix *.SQL* is appended to the filename (unless there is a period in the filename).

```
SQL>   INPUT
       1 SELECT EMPNO, ENAME, JOB
       2 FROM EMP
       3 WHERE JOB = 'ANALYST'
       4

SQL>   SAVE RESEARCH
```

- A file named ***RESEARCH.SQL*** is now in your directory.

Using the Host System's Text Editor
The EDIT Command

- You can run the host system's editor from ***SQL*Plus*** by entering the ***EDIT*** command:

```
SQL> EDIT
```

- ***EDIT*** edits the contents of the current buffer
- Any changes made during the edit session are saved back to the current buffer.
- To return to ***SQL*Plus,*** exit the system editor.
- To edit the contents of a file, enter ***EDIT*** followed by the name of the file:

```
SQL> EDIT RESEARCH
```

- The suffix ***.SQL*** is added to the filename (except in cases where the filename already contains a period)

Retrieving Stored Commands: The GET Command

- The *GET* command retrieves the contents of a disk file into the *SQL* buffer and displays it on the screen.

- To retrieve the file **RESEARCH** enter:

```
SQL> GET RESEARCH
```

- The *.SQL* suffix need not be specified.

- Now that the command has been brought back into the *SQL* buffer it can be edited, or executed with the *RUN* command.

Running Stored Commands: The START Command

- The ***START*** command retrieves a command file and runs the commands it contains.

```
SQL> START RESEARCH
```

- A command file executed with the ***START*** command may contain ***SQL*Plus*** commands and one or more ***SQL*** commands.

- As with the ***GET*** command, the ***.SQL*** suffix need not be specified.

Storing Output to a File: The SPOOL Command

- To store the results of a query in a file as well as display them on the screen use the ***SPOOL*** command:

```
SQL> SPOOL TRYFILE
```

- All information that is displayed on the screen will now be stored in the specified file as well.

- Unless there is a period in the filename, a default suffix will be appended to the filename to identify it as a listing file. This suffix is system dependent.

- To stop spooling to a file enter:

```
SQL> SPOOL OFF
```

- To print query results, spool them to a file as described above. Then instead of ***SPOOL OFF*** enter:

```
SQL> SPOOL OUT
```

- ***SPOOL OUT*** closes the listing file and prints the file to the system's default printer.

Creating Batch Reports

- To create a command file containing ***SQL*Plus*** commands and one or more ***SQL*** commands use the system editor.

```
SQL> EDIT TRYFILE
```

- Each ***SQL*** (not ***SQL*Plus***) command must be followed by a semi-colon (or a slash as the only character on the line following the ***SQL*** command).

```
SET ECHO OFF
SET AUTOCOMMIT ON
SET PAGESIZE 25
INSERT INTO EMP (EMPNO, ENAME, HIREDATE)
    VALUES (9999, 'GEIGER', SYSDATE);
INSERT INTO EMP (EMPNO, ENAME, DEPTNO)
    VALUES (3333, 'SAMSON', 20);
SPOOL NEW_EMP
SELECT * FROM EMP
    WHERE DEPTNO = 20
    OR DEPTNO IS NULL
/
SPOOL OFF
SET AUTOCOMMIT OFF
```

Now save the file and exit from the system editor.

- To execute this file use the ***START*** command.

You are ready to complete **Practice Session Four.**

6. Functions

Purpose of Functions

- ***Functions*** allow you to:
  - Modify values
  - Combine values to create new ones
  - Change value formats
- You can use them in any type of query, including the more complex queries that are discussed in the next chapter.

Functions: General Information

- *Like an expression*, a function can be used in these clauses:

  SELECT clause
  WHERE clause
  ORDER BY clause

- *Unlike an expression*, a function uses this format:

  FUNCTION(argument)

  It can also have multiple arguments:

  FUNCTION(arg 1, arg 2...arg N)

  e.g.

  SUBSTR (DNAME, 1, 4)

- The first argument of a function in ***SQL*** is always the value to which you are applying the function.

```
SQL> SELECT ENAME FROM EMP
        WHERE LENGTH (ENAME) = 6;
```

- There are many kinds of ***functions***

  Differentiated by Value Type

  ARITHMETIC Functions
  CHAR Functions
  DATE Functions

  Differentiated by Rows Affected

  INDIVIDUAL Functions
  Each Row is evaluated separate from other Rows

  GROUP Functions
  A Set of rows are evaluated collectively

- An advantage of a function:

  It can ***return*** a value based on multiple ***ARGUMENTS***

Common CHAR Functions

- Altogether, there are 15 ***CHAR Functions*** (described in the *SQL*Plus Reference Guide)*

- Here are some of the more common ones:

INITCAP*(ename)*
capitalizes first letter of each word

jack smith	*becomes*	Jack Smith
smith-jones	*becomes*	Smith-Jones
smith, jack	*becomes*	Smith, Jack

LENGTH*(ename)*
computes the number of characters in the string

length of ALLEN = 5
length of KING = 4

SUBSTR(job,1,4)
lists 4 characters starting with string's first character

CLERK	*becomes*	CLER
SALESMAN	*becomes*	SALE

Other CHAR Functions

LOWER
converts all characters in the string to lowercase

UPPER
converts all characters in the string to uppercase

LEAST
returns the value from a series of arguments that comes *first* alphabetically

GREATEST
returns the value from a series of arguments that comes *last* alphabetically

DATE Functions

- Specify ***DATE functions*** in the same manner as ***CHAR functions,*** with the function name followed by its arguments in ***( )***

- Here are three simple ***DATE functions:***

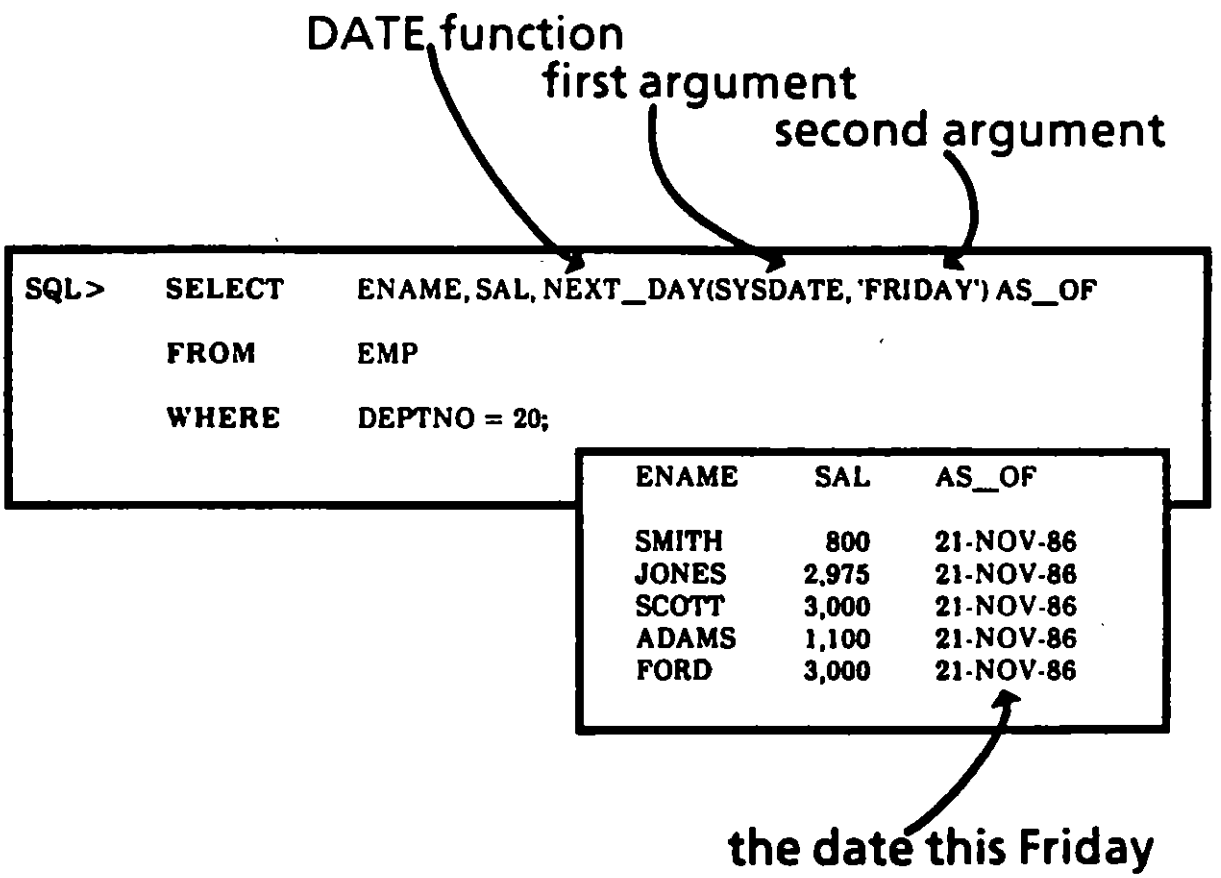
  ADD_MONTHS(hiredate,5***)***
  adds 5 months to *Hiredate*

  MONTHS_BETWEEN(sysdate, hiredate***)***
  calculates the number of months between *Hiredate* and *Sysdate*

  NEXT_DAY(hiredate, 'friday'***)***
  finds the date of the Friday after the employee was hired

Example of a DATE Function

Show what salaries are being paid in Dept 20 effective this coming Friday



The TO_CHAR Function

- Dates are displayed in the ORACLE default format unless you use the ***TO_CHAR*** function.
- The form is:
  TO_CHAR (date, date picture)
- The date will be represented as a character string according to the format of the date picture.

```
SQL> SELECT    ENAME, TO_CHAR(HIREDATE, 'Dy Mon dd, yyyy') HIRED
    FROM       EMP
    WHERE      DEPTNO = 10;
```

ENAME	HIRED
CLARK	Tue Jun 09, 1981
KING	Tue Nov 17, 1981
MILLER	Sat Jan 23, 1982

The TO_DATE Function

- The internal representation of a date is accurate to the second, and can range over many centuries.
- The **TO_DATE** function can convert a character string from a variety of formats to an ORACLE date data type.
- The form is:

  TO_DATE (character string, date picture)

  The character string will be converted to an ORACLE date according to the format of the date picture.

```
SQL> INSERT  INTO  EMP (EMPNO, ENAME, HIREDATE)
       VALUES  (7999, 'SAMS',
       TO_DATE ('070387083000', 'MMDDYYHHMISS'));
```

DATE Pictures

- Whenever you use a ***DATE picture*** you must enclose it in single quotes.
- Each ***DATE picture*** is composed of basic components.
- Some of these are:

Capitalization and Abbreviations
When you capitalize or abbreviate the picture, the output will reflect this:

Days		
dd	number	12
dy	abbreviated	fri
day	spelled out	friday
ddspth	spelled out, ordinal	twelfth
Month		
mm	number	03
mon	abbreviated	mar
month	spelled out	march
Year		
yy	year	87
yyyy	year and century	1987

Examples of DATE Pictures

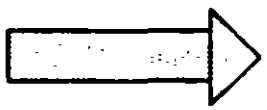
Example	*Picture*
Mar 12, 1987	***'Mon dd, yyyy'***
MAR 12, 1987	***'MON dd, yyyy'***
Thursday MARCH 12	***'Day MONTH dd'***
Mar 12 11:00am	***'Mon dd hh:miam'***
Thu, the twelfth	***'Dy, "the" ddspth'***

Numeric Functions

- Like ***CHAR*** and ***DATE*** functions, you specify ***numeric*** functions with the function name followed by its arguments in ( ).

- By using ***numeric functions***, you can perform calculations on a set of values in your table or view.

- In the following example, ***LEAST*** is an arithmetic function with two arguments (empno, mgr).



Which employees have a lower empno than their bosses'?

```
SQL>    SELECT    ENAME, EMPNO, MGR
                  LEAST(EMPNO,MGR) LOWNUM
        FROM      EMP
        WHERE     EMPNO < MGR;
```

ENAME	EMPNO	MGR	LOWNUM
SMITH	7369	7902	7369
ALLEN	7499	7698	7499
WARD	7521	7698	7521
JONES	7566	7839	7566
MARTIN	7654	7698	7654
BLAKE	7698	7839	7698
CLARK	7782	7839	7782

Other Numeric Functions

ABS (COMM - SAL)
Returns absolute value of the difference between COMM and SAL

GREATEST (SAL, COMM)
Returns the largest value (SAL or COMM)

ROUND (SAL, 0)
Rounds SAL to the closest dollar (with no decimal)

SIGN(COMM - SAL)

Sign =	- 1	if	COMM - SAL < 0
Sign =	0	if	COMM - SAL = 0
Sign =	+ 1	if	COMM - SAL > 0

TRUNC (SAL,0)
Truncates SAL to the closest dollar (always round down)

The Null Value Function: NVL

- Allows proper handling of **NULL** data values. (Without **NVL**, any number plus **NULL** evaluates to **NULL**.)
- The form is: **NVL (arg1, arg2)**
- **Arg1** is a column name. If **arg1** is not **NULL**, **NVL** returns its value.
- If **arg1** is **NULL**, **NVL** returns the value of **arg2**.

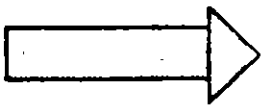
How much total compensation does each employee in Dept 30 receive?

```
SQL>    SELECT    ENAME, COMM +
                  SAL,
                  NVL(COMM,0) + SAL
        FROM      EMP
```

ENAME	COMM+SAL	NVL(COMM,0)+SAL
ALLEN	1,900	1,900
WARD	1,750	1,750
MARTIN	2,650	2,650
BLAKE		2,850
TURNER	1,500	1,500
JAMES		950

Since COMM is null for Blake and James, these values are also null.

GROUP Functions



Find the total amount of commissions paid to employees.

```
SQL>    SELECT    SUM(COMM)
        FROM      EMP;

SUM(COMM)
---------
     2,200
```

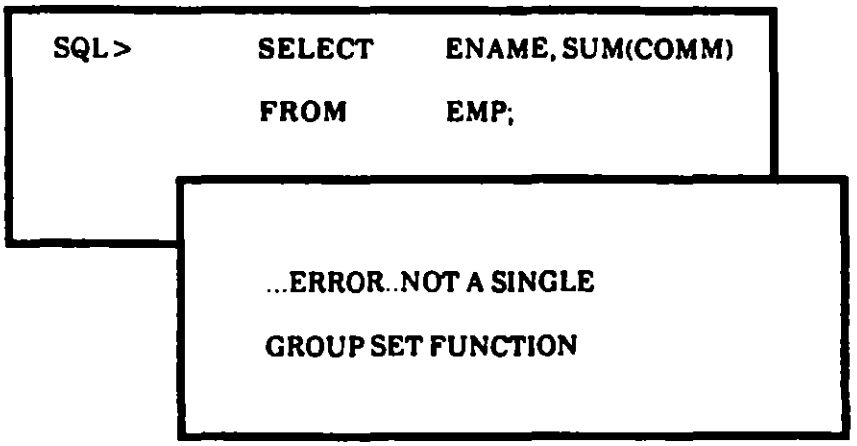
- In general, ***GROUP Functions***:

  Return a single value for a set of rows.

  Can be applied to *any* numeric values, and *some* ***CHAR*** and ***DATE*** values.

Selection Consistency

YOU CANNOT SELECT
INDIVIDUAL AND GROUP
RESULTS TOGETHER...

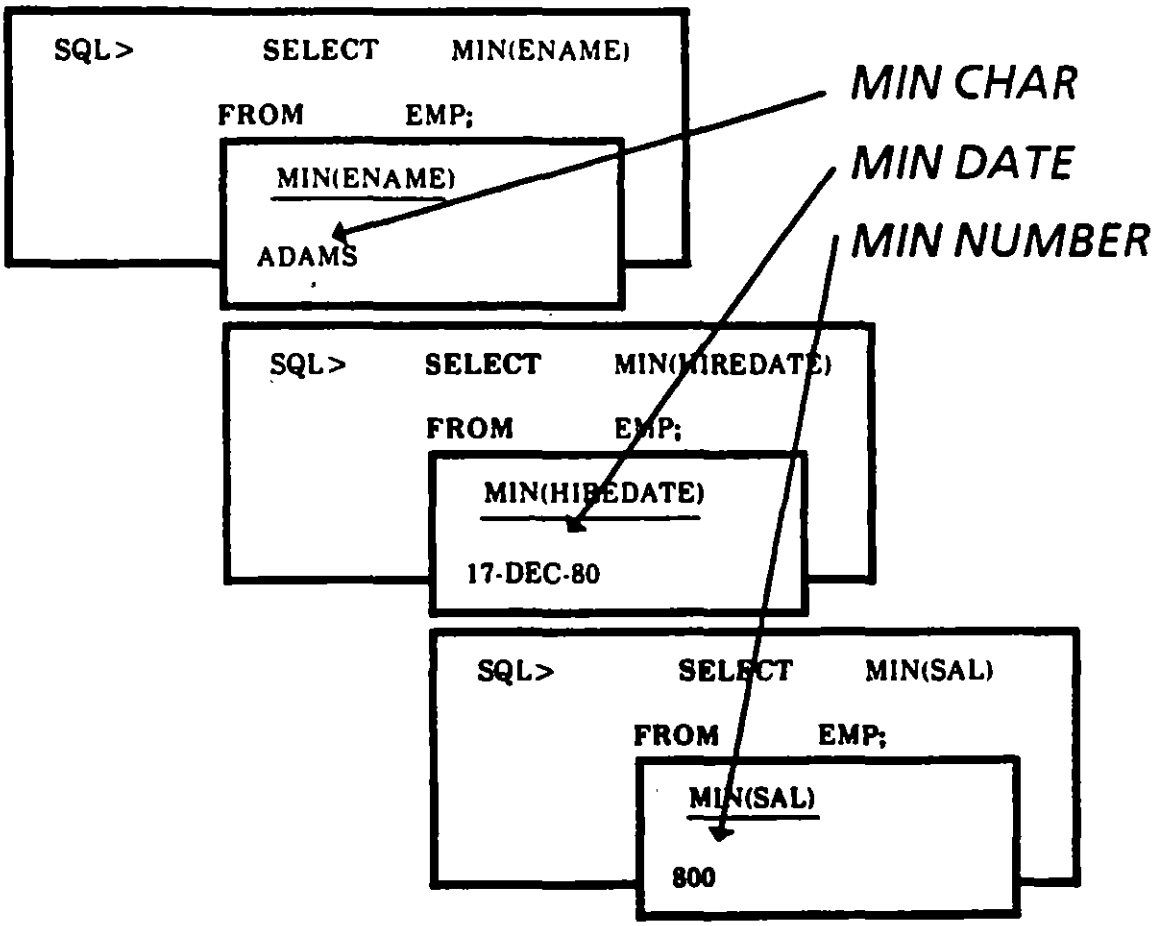


...*UNLESS* YOU CONSTRUCT
A VALID ***GROUP BY*** CLAUSE

Note: The GROUP BY clause will be discussed later in the chapter.

Group Functions for CHAR, DATE, or Number

- ***MIN, MAX,*** and ***COUNT*** can be used with any value type:



- ***MAX*** and ***COUNT*** work the same way

COUNT DISTINCT

How many people have jobs?

```
SQL>        SELECT   COUNT(JOB)
            FROM     EMP;

  COUNT(JOB)
  ----------
          14
```

How many different types of jobs are there?

```
SQL>        SELECT   COUNT(DISTINCT JOB)
            FROM     EMP;

  COUNT(DISTINCT JOB)
  -------------------
                    5
```

- ***COUNT DISTINCT*** returns the number of unique values in a column.

More GROUP Functions (for Numeric values only)

- **AVG** → *Find the average salary of employees.*

```
SQL>    SELECT  AVG(SAL)
        FROM    EMP;

        AVG(SAL)
        2,073.21
```

- **STDDEV** → *Find the standard deviation of employee salaries.*

```
SQL>    SELECT
        STDDEV(SAL)

        STDDEV(SAL)
        1,182.50
```

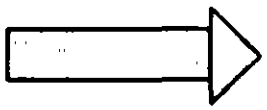
- **SUM** → *Find the total amount of salaries paid to employees.*

```
SQL> SELECT  SUM(SAL)
     FROM    EMP;

     SUM(SAL)
     29,025.00
```

The GROUP BY clause

- You use the ***GROUP BY*** clause to define multiple groups of rows.
- Every member of the group has at least one value in common.
- You specify the column(s) containing these common values in the ***GROUP BY*** clause.



What is the total and average salary for each department?

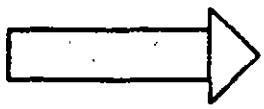
```
SQL> SELECT     DEPTNO, SUM(SAL), AVG(SAL)
     FROM       EMP
     GROUP BY   DEPTNO;
```

DEPTNO	SUM(SAL)	AVG(SAL)
10	8,750.00	2,916.67
20	10,875.00	2,175.00
30	9,400.00	1,566.67

- Notice that a query with a ***GROUP BY*** clause returns one row per group.

Multiple Criteria for Grouping Rows

- Depending on your question, you may need to group by more than one column.

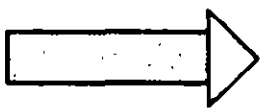


How many employees are there in each department?

```
SQL> SELECT   DEPTNO, COUNT(*)
     FROM     EMP
     GROUP BY DEPTNO;
```

DEPTNO	COUNT(*)
10	3
20	5
30	6

- In the example above you only group by one column; the query below entails the grouping of two columns:



How many employees are there in each job category in each department?

```
SQL> SELECT   DEPTNO, JOB, COUNT(*)
     FROM     EMP
     GROUP BY DEPTNO, JOB;
```

DEPTNO	JOB	COUNT(*)
10	CLERK	1
10	MANAGER	1
10	PRESIDENT	1
20	ANALYST	2
20	CLERK	2
20	MANAGER	1
30	CLERK	1
30	MANAGER	1
30	SALESMAN	4

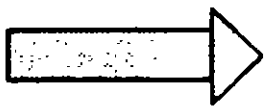
Selecting Specific Groups

- The ***HAVING*** clause allows you to select groups that meet specific condition(s).

- It is analogous to the ***WHERE*** clause, but it serves a different purpose:

  The ***WHERE*** clause places conditions on the ***SELECT*** clause

  The ***HAVING*** clause places conditions on the ***GROUP BY*** clause



Which departments have a payroll exceeding $9,000? (Do not include commissions.)

```
SQL> SELECT     DEPTNO, SUM(SAL)
     FROM       EMP
     GROUP BY   DEPTNO
     HAVING     SUM(SAL) > 9000;
```

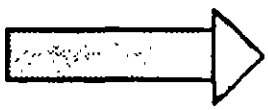
DEPTNO	SUM(SAL)
20	10,875
30	9,400

Syntax Review

- The complete syntax for a query is as follows:

SELECT	col 1, col 2...col n
FROM	table 1, table 2...table n (these can also be ***VIEWS***)
WHERE	col conditions are true
GROUP BY	col 1, col 2...col n
HAVING	group conditions true
ORDER BY	col 1, col 2...col n

- Here's a query that uses these clauses together:



Except for clerical help, which departments have a payroll exceeding $8,000? (Do not include commissions, and list the department with the lowest amount first.)

```
SQL> SELECT    DEPTNO, SUM(SAL)
     FROM      EMP
     WHERE     JOB != 'CLERK'
     GROUP BY  DEPTNO
     HAVING    SUM(SAL) > 8000
     ORDER BY  SUM(SAL);
```

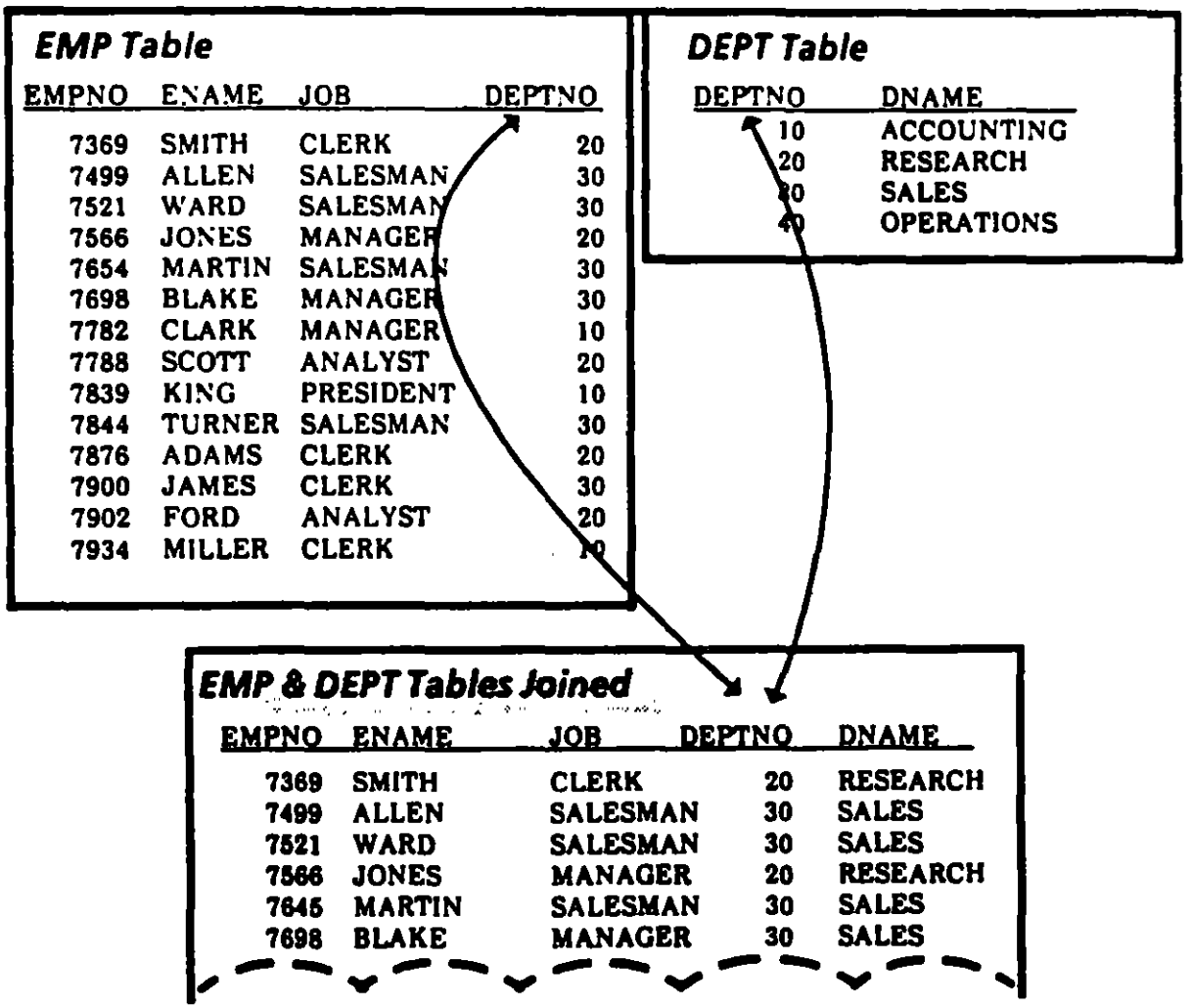
DEPTNO	SUM(SAL)
30	8,450
20	8,975

You are ready to complete **Practice Session Five.**

7. Advanced Queries

Joins
The Basic Concept

- You use *Joins* to combine *columns* from different tables



EMP Table

EMPNO	ENAME	JOB	DEPTNO
7369	SMITH	CLERK	20
7499	ALLEN	SALESMAN	30
7521	WARD	SALESMAN	30
7566	JONES	MANAGER	20
7654	MARTIN	SALESMAN	30
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7788	SCOTT	ANALYST	20
7839	KING	PRESIDENT	10
7844	TURNER	SALESMAN	30
7876	ADAMS	CLERK	20
7900	JAMES	CLERK	30
7902	FORD	ANALYST	20
7934	MILLER	CLERK	10

DEPT Table

DEPTNO	DNAME
10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

EMP & DEPT Tables Joined

EMPNO	ENAME	JOB	DEPTNO	DNAME
7369	SMITH	CLERK	20	RESEARCH
7499	ALLEN	SALESMAN	30	SALES
7521	WARD	SALESMAN	30	SALES
7566	JONES	MANAGER	20	RESEARCH
7645	MARTIN	SALESMAN	30	SALES
7698	BLAKE	MANAGER	30	SALES

How to do a Simple Join (Equijoin)

- Table prefixes prevent ambiguity

```
SQL>    SELECT    EMPNO, ENAME, JOB,
                  EMP.DEPTNO, DNAME
        FROM      EMP, DEPT
        WHERE     EMP.DEPTNO = DEPT.DEPTNO;
```

EMP & DEPT Tables Joined

EMPNO	ENAME	JOB	DEPTNO	DNAME
7369	SMITH	CLERK	20	RESEARCH
7499	ALLEN	SALESMAN	30	SALES
7521	WARD	SALESMAN	30	SALES
7566	JONES	MANAGER	20	RESEARCH
7645	MARTIN	SALESMAN	30	SALES
7698	BLAKE	MANAGER	30	SALES

- Specifies which columns are being joined

What happens when you forget the WHERE clause?

```
SQL>    SELECT    ENAME, EMP.DEPTNO,LOC
        FROM      EMP, DEPT;
```

- Each row of one table matches every row of the other table

- This results in a ***Cartesian Product***

ENAME	DEPTNO	LOC
SMITH	20	NEW YORK
ALLEN	30	NEW YORK
WARD	30	NEW YORK
JONES	20	NEW YORK
MARTIN	30	NEW YORK
BLAKE	30	NEW YORK
CLARK	10	NEW YORK
SCOTT	20	NEW YORK
KING	10	NEW YORK
TURNER	30	NEW YORK
ADAMS	20	NEW YORK
JAMES	30	NEW YORK
FORD	20	NEW YORK
MILLER	10	NEW YORK
SMITH	20	DALLAS
ALLEN	30	DALLAS
WARD	30	DALLAS
JONES	20	DALLAS
MARTIN	30	DALLAS
BLAKE	30	DALLAS
CLARK	10	DALLAS
SCOTT	20	DALLAS
KING	10	DALLAS
TURNER	30	DALLAS
ADAMS	20	DALLAS
JAMES	30	DALLAS
FORD	20	DALLAS
MILLER	10	DALLAS

Outer Joins

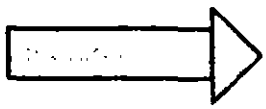
```
SQL>    SELECT    ENAME, DEPT.DEPTNO,LOC
        FROM      EMP, DEPT
        WHERE     EMP.DEPTNO(+) = DEPT.DEPTNO;
```

ENAME	DEPTNO	LOC
CLARK	10	NEW YORK
MILLER	10	NEW YORK
KING	10	NEW YORK
SMITH	20	DALLAS
SCOTT	20	DALLAS
JONES	20	DALLAS
ADAMS	20	DALLAS
FORD	20	DALLAS
ALLEN	30	CHICAGO
BLAKE	30	CHICAGO
TURNER	30	CHICAGO
JAMES	30	CHICAGO
MARTIN	30	CHICAGO
WARD	30	CHICAGO
	40	BOSTON

- If there are any values in DEPT.DEPTNO that don't have corresponding values in EMP.DEPTNO, (such as Dept 40), then the ***Outer Join*** symbol ***(+)*** will cause a NULL value to appear in the joined table.

Self-Join

- When you want to join one row in a table with another row in the same table, you perform a ***Self-Join.***



Determine the name of each employee's manager.

```
SQL>  SELECT    WORKER.ENAME, MANAGER.ENAME MANAGER
      FROM      EMP WORKER, EMP MANAGER
      WHERE     WORKER.MGR = MANAGER.EMPNO;
```

- You join a table to itself, as though it were two separate tables.
- It is necessary to use an alias for at least one occurrence of the table in the ***FROM*** clause, to distinguish the column names from one another.

EMP Table

ENAME	MANAGER
SCOTT	JONES
FORD	JONES
ALLEN	BLAKE
WARD	BLAKE
MARTIN	BLAKE
TURNER	BLAKE
JAMES	BLAKE
MILLER	CLARK
ADAMS	SCOTT
JONES	KING
BLAKE	KING
CLARK	KING
SMITH	FORD

Non-Equijoin

- If the join condition in the ***WHERE*** clause specifies an equivalency (=), it is an ***Equijoin.***
- A join condition with any other operator is a ***Non-Equijoin.***

* *Introducing a new table: SALGRADE*

SALGRADE Table

GRADE	LOSAL	HISAL
1	700	1,200
2	1,201	1,400
3	1,401	2,000
4	2,001	3,000
5	3,001	9,999

Whose salary falls in Grade Level 3?

```
SQL>    SELECT    ENAME, SAL
        FROM      EMP, SALGRADE
        WHERE     GRADE = 3
        AND       SAL BETWEEN LOSAL AND HISAL;
```

ENAME	SAL
ALLEN	1,600
TURNER	1,500

Set Operators:

Joining Rows

- *Set Operators* combine 2 or more queries into one result.

UNION - set union Rows of first query plus rows of second query, less duplicate rows
INTERSECT - set intersection Rows both queries have in common
MINUS - set difference Rows unique to the first query

Introducing New Views...

* *Here are 3 views of the EMP table that we'll use for some of the upcoming examples:*

ACCOUNT View

ENAME	SAL	JOB
CLARK	2,450	MANAGER
KING	5,000	PRESIDENT
MILLER	1,300	CLERK

SALES View

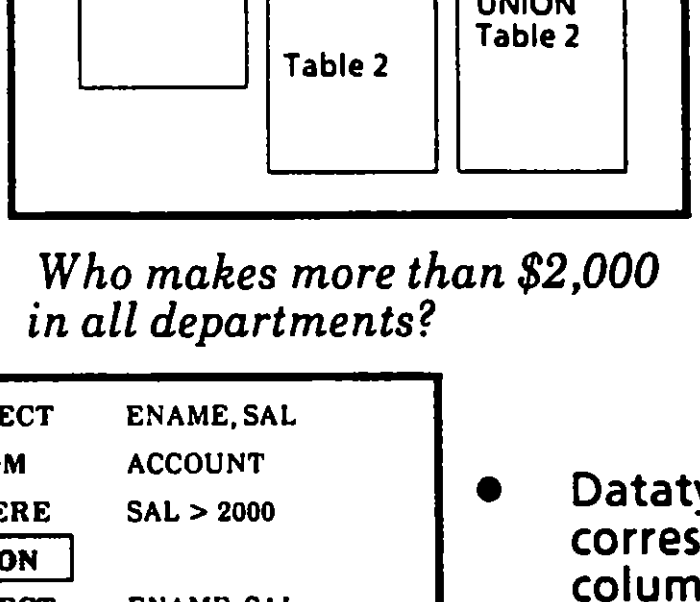
ENAME	SAL	JOB
ALLEN	1,600	SALESMAN
WARD	1,250	SALESMAN
MARTIN	1,250	SALESMAN
BLAKE	2,850	MANAGER
TURNER	1,500	SALESMAN
JAMES	950	CLERK

RESEARCH View

ENAME	SAL	JOB
SMITH	800	CLERK
JONES	2,975	MANAGER
SCOTT	3,000	ANALYST
ADAMS	1,100	CLERK
FORD	3,000	ANALYST

UNION Operator

- ***UNION*** returns all distinct rows selected by either of the queries to which it applies
- It combines rows from multiple tables or views



Who makes more than $2,000 in all departments?

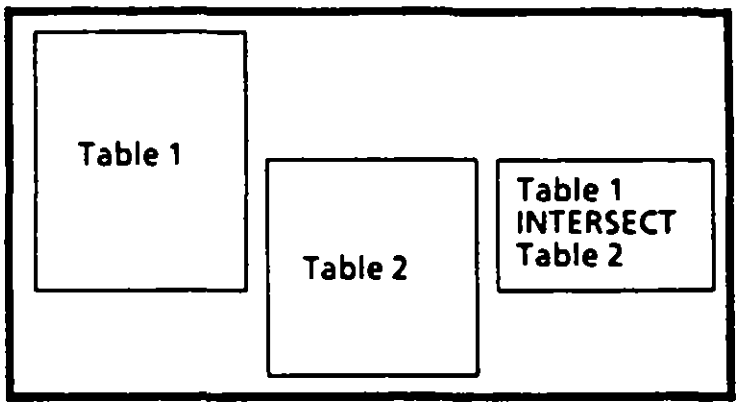
```
SQL> SELECT   ENAME, SAL
     FROM     ACCOUNT
     WHERE    SAL > 2000
     UNION
     SELECT   ENAME, SAL
     FROM     RESEARCH
     WHERE    SAL > 2000
     UNION
     SELECT   ENAME, SAL
     FROM     SALES
     WHERE    SAL > 2000;
```

- Datatype of corresponding columns *must* be the same

ENAME	SAL
BLAKE	2,850
CLARK	2,450
FORD	3,000
JONES	2,975
KING	5,000
SCOTT	3,000

INTERSECT Operator

- ***INTERSECT*** finds common values in multiple tables.
- Columns must be of the same datatype



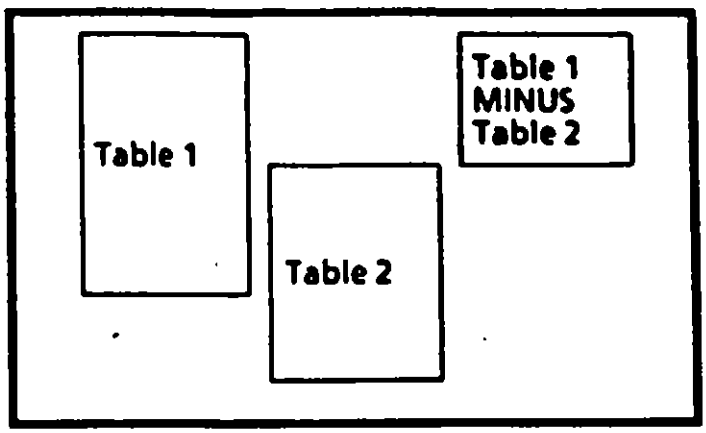
What jobs do all departments have in common?

```
SQL>  SELECT   JOB
      FROM     ACCOUNT
      INTERSECT
      SELECT   JOB
      FROM     RESEARCH
      INTERSECT
      SELECT   JOB
      FROM     SALES;
```

```
JOB
---------
CLERK
MANAGER
```

MINUS Operator

- ***MINUS*** returns all rows selected by the *preceding* query, but not the *following* query



Are there any jobs in the Accounting Department that you don't find in the Sales Department?

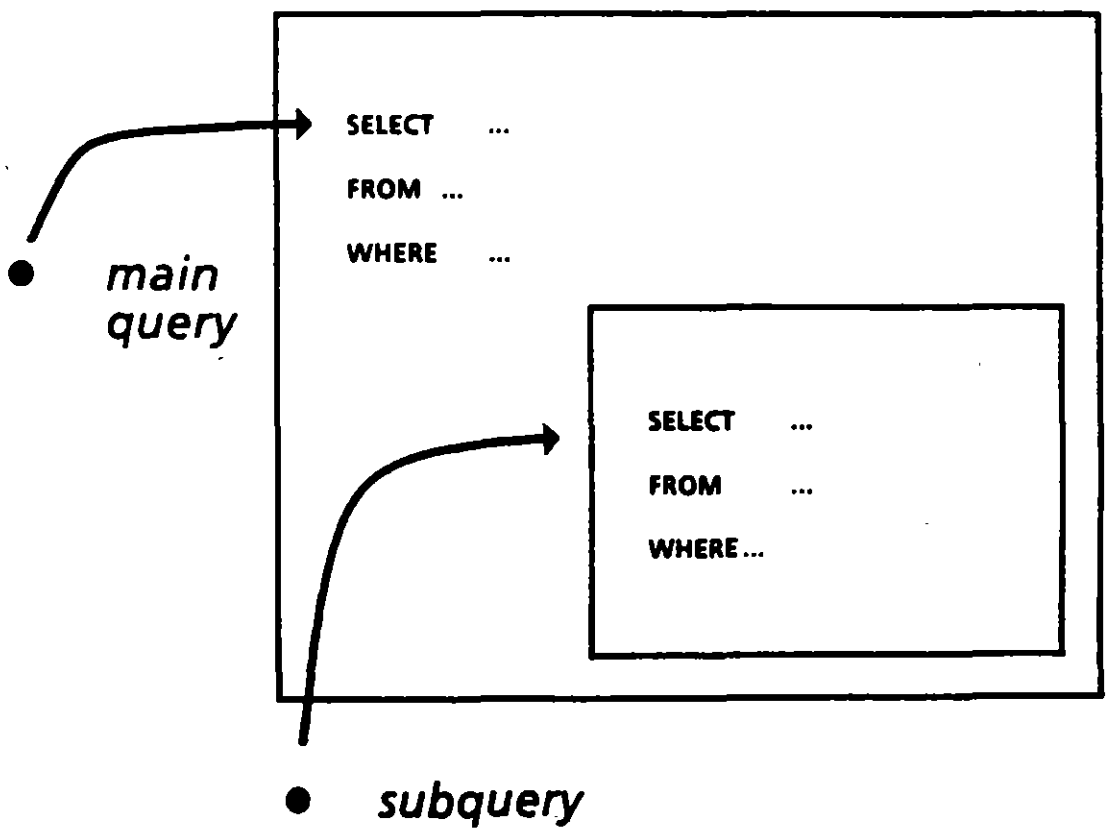
```
SQL>  SELECT  JOB
      FROM    ACCOUNT
      MINUS
      SELECT  JOB
      FROM    SALES;

      JOB
      ---------
      PRESIDENT
```

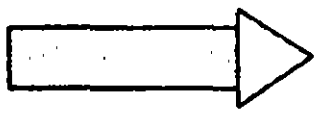
Introduction to Subqueries

- A ***subquery*** (nested query) is a query contained in a ***WHERE*** clause.

- The results are used in solving the main query



An Example of Setting up a Subquery



Which employees work in Smith's department?

STRATEGY

*Step 1 - **SUBQUERY***
Figure out which department Smith works in

*Step 2 - **MAIN QUERY***
Select all employees who work in that department

```
SQL>    SELECT    ENAME, DEPTNO
        FROM      EMP
        WHERE     DEPTNO =
                  (SELECT    DEPTNO
                  FROM       EMP
                  WHERE      ENAME = 'SMITH');
```

ENAME	DEPTNO
SMITH	

- Determined in the ***subquery***
- Determined in the ***main query***

Subqueries can have Multiple Layers

- Nesting may continue indefinitely
- The subquery can access tables not used by the main query
- A subquery cannot have an ORDER BY clause

Multiple Subqueries

- A *subquery* may occur as one half of a relational operator

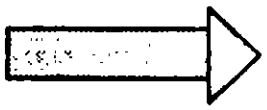
Which employees have the same job as Clark or have a salary greater than his?

```
SQL>  SELECT  ENAME, JOB, SAL
      FROM    EMP
      WHERE   JOB =
              (SELECT   JOB
              FROM      EMP
              WHERE     ENAME = 'CLARK')
      OR      SAL >
              (SELECT   SAL
              FROM      EMP
              WHERE     ENAME = 'CLARK');
```

ENAME	JOB	SAL
JONES	MANAGER	2,975
BLAKE	MANAGER	2,850
CLARK	MANAGER	2,450
SCOTT	ANALYST	3,000
KING	PRESIDENT	5,000
FORD	ANALYST	3,000

Multiple Tables and Subqueries

- You can retrieve information from more than one table while doing a ***subquery.***



Which employees in New York have a salary greater than Scott ?

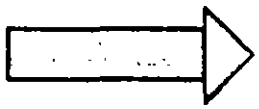
```
SQL>  SELECT   ENAME, JOB, SAL
      FROM     EMP, DEPT
      WHERE    LOC = 'NEW YORK'
      AND      EMP.DEPTNO = DEPT.DEPTNO
      AND      SAL >
               (SELECT   SAL
               FROM      EMP
               WHERE     ENAME = 'SCOTT');
```

ENAME	JOB	SAL
KING	PRESIDENT	5,000

How would you rewrite this query to specify SCOTT who works in DALLAS? (There may be more than one SCOTT in the company.)

Group Functions in a Subquery

- If you select a regular column and a group function together, you get an error message.



Which employee was hired first?

```
SQL>    SELECT    ENAME, MIN(HIREDATE)
        FROM      EMP;

...ERROR...NOT A SINGLE
GROUP SET FUNCTION...
```

- As an alternative, you can perform the group function in a subquery

```
SQL>    SELECT    ENAME, HIREDATE
        FROM      EMP
        WHERE     HIREDATE =
                  (SELECT    MIN(HIREDATE)
                  FROM       EMP);

ENAME      HIREDATE
SMITH      17-DEC-80
```

Smith was hired first

You are ready to complete **Practice Session Six.**

Class Notes

April 1988

Introduction to ORACLE for Developers

Addendum

ORACLE®

Introduction to ORACLE for Developers Class Notes Addendum

SQL*Plus Author: Mark Rosen
SQL*Plus Reviewers: Glynn Durham, Judith Vandenberg
SQL*Loader Author: Simmie Kastner
SQL*Loader Reviewers: Rudy Corsi, Glynn Durham, Donald Feinberg, Carol Zimmerman

Table of Contents

1. Grants in SQL*Plus

Data Security Overview

GRANT

System privileges

Table privileges

View privileges

REVOKE

Revoking privileges

System Privileges

3 levels of system privileges:

DBA

all privileges

RESOURCE

logon, create new tables

CONNECT

logon, query only

Only a DBA has authority to grant new userids.

```
SQL> GRANT CONNECT TO SCOTT
     IDENTIFIED BY TIGER;
```

You can change your password.

```
SQL> GRANT CONNECT TO SCOTT
     IDENTIFIED BY TIGER;
```

Table Privileges

Two ways to gain table access:

Create the table yourself

Obtain permission from the table's owner:

```
SQL> GRANT SELECT, INSERT
     ON EMP
     TO SCOTT;
```

These privileges can be granted:

SELECT	INSERT	UPDATE
DELETE	ALTER	INDEX

To give someone all privileges:

```
SQL> GRANT ALL
     ON EMP
     TO SCOTT;
```

Synonyms

If another user has granted you SELECT privileges, here is how you would gain access:

```
SQL> SELECT *
     FROM SCOTT.EMP;
```

Userid of table's owner

As an alternative, you can create a synonym, using a name that is more meaningful to you.

Here's how you create a synonym called EMPLOYEE for Allen's table called EMP:

You can now query EMPLOYEE like other tables.

```
SQL> CREATE SYNONYM.EMPLOYEE
     FOR ALLEN.EMP;
```

More About Table Privileges

When accessing someone else's table, you may get a message saying "Table or View does not exist." This means:

You don't have access to the table or view.

or

The table or view really does not exist.

You can also give others the right to grant privileges for one of your tables:

```
SQL> GRANT ALL
     ON EMP
     TO SCOTT
     WITH GRANT OPTION;
```

Revoking Privileges

System privileges

Can only be revoked by the DBA.

Table privileges

May be revoked at any time.

```
SQL> REVOKE INSERT
     ON EMP
     FROM SCOTT;
```

2. Indexes in SQL*Plus

Indexing Tables

Purpose of an index

It helps the RDBMS query large tables more quickly.

Rather than searching the entire table, it navigates more efficiently.

To create an index:

```
SQL> CREATE INDEX EMP_ENAME
     ON EMP (ENAME);
```

To drop an index:

```
SQL> DROP INDEX EMP_ENAME;
```

Information About Indexing

- Index only large tables (at least 50 rows).
- Insert data before indexing.
- A table may have any number of indexes.
- Typically, you should index the column that uniquely identifies the row (the primary key).
- There is no impact on the SQL syntax.
- Indexes are automatically updated.

Using Indexes to Ensure Unique Values

In addition to improving performance, you can use indexes to ensure that each value in a column is unique.

Example: Make sure that each employee number in the EMP table appears only once.

```
SQL> CREATE UNIQUE INDEX EMP_EMPNO
     ON EMP (EMPNO);

INDEX CREATED.
```

Once you create a unique index, you will get an error message anytime you try to insert or update a row with the same EMPNO as some other existing row.

3. SQL*Loader

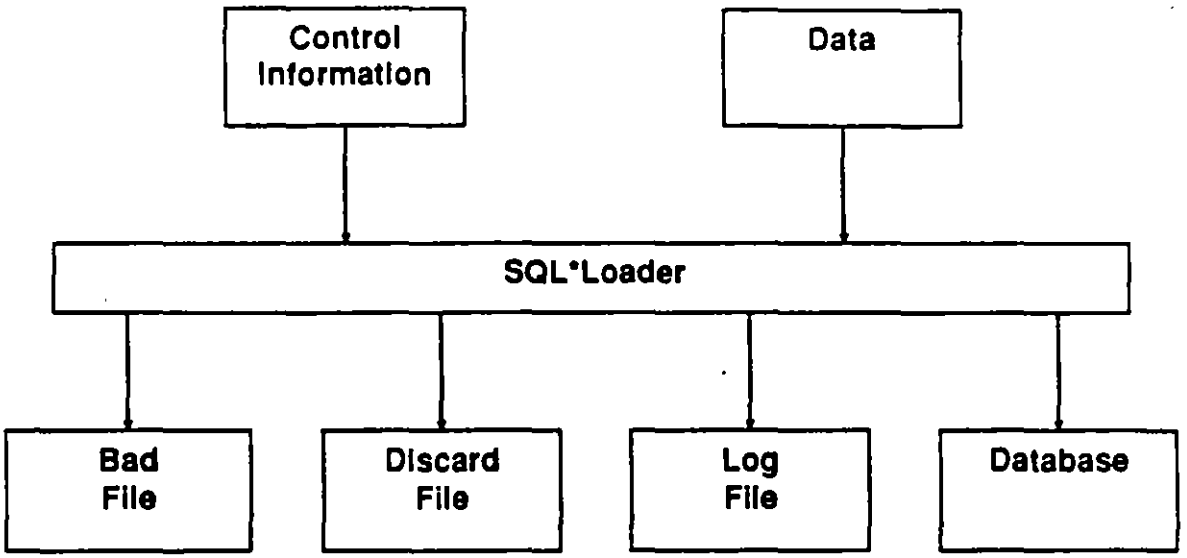
SQL*Loader Features

- DB2 compatibility
- Ability to load from one or many source files
- Fixed format, delimited format, and variable length records
- Disk or tape input
- Character or binary data

SQL*Loader Features

- Ability to load into multiple tables
- A logical record can be one or more physical records
- A single physical record can hold multiple logical records
- Automatically generated sequence numbers
- Filtering of data

SQL*Loader Operation



Control File

- Names of the data files
- Format of the data files
- Specifications for loading data
- Data to be loaded (optional)

Loader Output Files

Log File

Header

Global information

Table information

Data file information

Table load information

Summary statistics

Bad File

Rejected records, in format loadable by the control file

Discard File

Records which do not meet control file specifications, in format loadable by the control file

Control Files:
The LOAD DATA Command

```
LOAD DATA
     {INFILE | INDDN} {filename | *}
     [ {BADFILE | BADDN} filename]
     [ {DISCARDFILE | DISCARDDN} filename]
     [ {DISCARDS | DISCARDMAX} n ]
     [ APPEND | REPLACE | INSERT ]

INTO TABLE tablename
     [ FIELDS delimiter spec]

     ( columnname [ columnspecification]
       [ ,columnname [columnspecification] ... ] )

[ INTO TABLE tablename ... ]

[ BEGINDATA
  data ]
```

The LOAD DATA Command

The bad file is always created.

The default bad file name is **controlfilename.BAD**.

The discard file is only created if specified here or in the command line options (discussed later).

The discard file is produced explicitly by naming it, or implicitly by specifying maximum discards.

The default discard file name is **controlfilename.DSC**.

If "*" is specified for INFILE, BEGINDATA must be used.

Multiple tables can be specified.

The LOAD DATA Command

APPEND

Allows rows to be added to a table

INSERT

Allows rows to be placed in an empty table only

REPLACE

Overwrites rows in a table

Data Delimiters

Specifying termination characters

TERMINATED BY [WHITESPACE | [X] 'char']

```
TERMINATED BY ','
```

Specifying enclosing characters

[OPTIONALLY] ENCLOSED BY [X] 'char'

```
ENCLOSED BY '"'
```

A Variable Field Length Example

```
LOAD DATA
INFILE * APPEND
DISCARDFILE CASE1
INTO TABLE DEPT
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(DEPTNO,DNAME,LOC)

BEGINDATA
12,research,"saratoga"
14,"accounting",cleveland
11,"art",salem
13,finance,"boston"
21,"sales",phila.
22,"sales",rochester
42,"int'l","san fran"
```

```
SQL> SELECT * FROM DEPT;

    DEPTNO DNAME      LOC
---------- ---------- ----------
         (...old rows...)

        12 research   saratoga
        14 accounting cleveland
        11 art        salem
        13 finance    boston
        21 sales      phila.
        22 sales      rochester
        42 int'l      san fran
```

FILE READ Mode

```
LOAD DATA
     {INFILE | INDDN} {filename | *}

     [ STREAM | RECORD | FIXED len [ BLOCKSIZE size ] |
       VARIABLE [ len ] ]

     [ {BADFILE | BADDN} filename]
     [ {DISCARDFILE | DISCARDDN} filename]
     [ {DISCARDS | DISCARDMAX} n ]
     [  APPEND | REPLACE | INSERT ]

INTO TABLE tablename
     [  FIELDS delimiter spec]

     ( columnname  [ columnspecification]
       [ ,columnname  [columnspecification] ... ]  )

[ INTO TABLE tablename ... ]

[ BEGINDATA
  data ]
```

Note: BEGINDATA will work only if RECORD is specified.

FILE READ Mode

STREAM

The data file is treated as a sequential byte stream.

RECORD

The default. SQL*Loader uses the operating system file management. If none exists, the effect is the same as STREAM.

FIXED

Records are of fixed length as specified by **LEN**.

VARIABLE

Records are variable in length. The length is contained in the first two bytes. **LEN** is a maximum for buffer optimization.

Invoking the Loader

SQLLOAD username/password filename <filename> <options>

Options:

Can be on the command line.

Can be embedded in the control file.

Can include:

DISCARDMAX

SKIP

LOAD

ERRORS

SILENT

Datatypes

SQL*Loader datatypes include:

CHAR

DATE

INTEGER [EXTERNAL]

FLOAT [EXTERNAL]

DECIMAL [EXTERNAL]

Datatype Example

```
...
INTO TABLE DEPT
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(DEPTNO INTEGER ,DNAME CHAR (10)  ,LOC CHAR (10))
```

DATE Dataype

To load a date in default SQL format, the data type can be INTEGER EXTERNAL, as follows:

...(HIREDATE INTEGER EXTERNAL)...

The date data would be in the form DD-MON-YY:

...12-NOV-85...

To load a date in some other format, the DATE datatype must be used with a date mask, as follows:

...(HIREDATE DATE "DD-Month-YYYY") ...

The date data can now be in the form:

...12-November-1985...

Specifying Fixed Positions

(fieldname POSITION (start:end) [datatype]

[, fieldname POSITION(start:end) [datatype] ...])

Fixed Position Example

```
LOAD DATA
INFILE CASE2.DAT
APPEND
INTO TABLE EMP
( EMPNO       POSITION(01:04)   INTEGER EXTERNAL,
  ENAME       POSITION(06:15)   CHAR,
  JOB         POSITION(16:25)   CHAR,
  MGR         POSITION(27:30)   INTEGER EXTERNAL,
  SAL         POSITION(32:39)   DECIMAL EXTERNAL,
  COMM        POSITION(41:48)   DECIMAL EXTERNAL,
  DEPTNO      POSITION(50:51)   INTEGER EXTERNAL)
```

Advanced Loading:
Multiple Physical Records

CONTINUEIF { THIS | NEXT | LAST } condition

THIS

If the condition is true in this record, concatenate the next physical record to it; continue until the condition is false.

If the condition is false, then this is the last physical record of the logical record.

NEXT

If the condition is true in the next physical record, concatenate it to this record, and continue until the condition is false in the next physical record.

If the condition is false in the next physical record, then this is the last physical record of the logical record.

LAST

The same as THIS, except always test against the last non-blank character.

Multiple Physical Record Example

For the following CONTINUEIF clauses:

```
CONTINUEIF THIS (1:2)='%%'
```

```
CONTINUEIF NEXT (1:2)='%%'
```

and the following data:

```
%%aaaaaaaa
%%bbbbbbbb
  cccccccc
%%ddddddddd
```

```
  aaaaaaaa
%%bbbbbbbb
%%cccccccc
  ddddddddd
```

the logical record would be the same in both cases:

```
aaaaaaaabbbbbbbbcccccccc
```

Multiple Physical Record Example

```
LOAD DATA
INFILE CASE3.DAT
REPLACE
CONTINUEIF THIS (1) = '*'
INTO TABLE EMP

(EMPNO       POSITION(01:04) INTEGER EXTERNAL,
 ENAME       POSITION(07:16) CHAR,
 JOB         POSITION(18:26) CHAR,
 MGR         POSITION(28:31) INTEGER EXTERNAL,
 SAL         POSITION(33:40) DECIMAL EXTERNAL,
 COMM        POSITION(42:49) INTEGER EXTERNAL,
 DEPTNO      POSITION(51:52) INTEGER EXTERNAL,
 HIREDATE    POSITION(53:61) INTEGER EXTERNAL)
```

Note that position number 1 is specified in both the CONTINUEIF and the datatype specification.

Data

```
*9982  CLARK      MA
 NAGER   7839     2572          2512-NOV-85
*9938  KING       PR
 ESIDENT       5500.00      -10 2505-APR-83
*9499  ALLEN      SA
 LESMAN  7698  1600.00   300.00 25 3-JUN-84
```

Automatic Sequencing

(...columnname SEQUENCE({n | MAX | COUNT} [,increment]) ...)

n

Start with integer value n.

MAX

Start with the maximum column value plus the increment.

COUNT

Start with the number of rows already in the table, plus the increment.

increment

Increment by this amount for each successive row. The default increment is 1.

```
(...empno SEQUENCE(MAX,2)...)
```

Filtering

After the table name, use:

```
WHEN   { (start [ :end ] ) | column_name }
             operator
       { 'char string' | X'hex string' }

       [ AND ... ]
```

Operators are equal or not equal.

Filtering Examples

```
WHEN (5) = 'q'
```

```
WHEN (DEPTNO= '10') AND (JOB = 'SALES')
```

Comparisons with external data types are expressed as character strings.

A Filtering LOAD DATA Command

```
LOAD DATA
INFILE CASE5.DAT
APPEND
INTO TABLE EMP
WHEN DEPTNO = '10'
( EMPNO       POSITION(01:04)   INTEGER EXTERNAL,
  ENAME       POSITION(06:15)   CHAR,
  JOB         POSITION(16:25)   CHAR,
  MGR         POSITION(27:30)   INTEGER EXTERNAL,
  SAL         POSITION(32:39)   DECIMAL EXTERNAL,
  COMM        POSITION(41:48)   DECIMAL EXTERNAL,
  DEPTNO      POSITION(50:51)   INTEGER EXTERNAL)
```

Data to be Filtered

```
9782 CLARK      MANAGER   7839   100.50              10
9934 MILLER     CLERK     7782                       10
9566 JONES      MANAGER   7839   100.75              20
9499 ALLAN      SALESMAN  7698              300.00   30
9654 MARTIN     SALESMAN  7698   100.50    1400.00   30
9658 CHAN       ANALYST   7566   100.00              20
```

A portion of the resulting log file:

```
Table EMP:
  2 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  4 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Space allocated for bind array:     3840 bytes (64 rows)
Space otherwise allocated:          6740 bytes

Total logical records skipped:          0
Total logical records read:             6
Total logical records rejected:         0
Total logical records discarded:        4
```

Practice Session One

1) Log onto *SQL*Plus*.

2) Retrieve all data from ***EMP***.

3) Display the names of all employees who work in department 20. Make sure the column containing employee names is called EMPLOYEE.

4) Display the names of all employees in department 10 and 20, ordering the listing in descending alphabetical order.

5) Display all department locations that do not start with the letter 'N'.

Challenge Exercises

6) Do any employees have a salary that is more than their commission? If so, present the listing with salaries in descending order. If two or more employees have the same salary, order them alphabetically by name.

Practice Session One (continued)

7) Prepare a report containing employee name, commission percent and length of service (in weeks). Ensure that the report does not have any acronyms or abbreviations. Order the listing by employee names. Do not include employees who are not eligible for a commission.

Practice Session Two

1) Add Joe Wilson to your ***EMP*** table. Make up information about him to complete your record.

2) Promote Joe to Manager.

3) Joe was fired. Delete his record from the table.

Challenge Exercises

4) Give all employees in the Chicago office (department 30) a 10% raise, and modify the appropriate table to reflect that.

6) Create a Facilities department for Hitech Inc. and assign it number 50. No other information is known yet about this department.

Practice Session Three

1) Add a column to the ***EMP*** table to store the first name of each employee.

2) Change ***EMP*** so that it can accommodate last names that are 24 characters long.

3) Create a view of all employees in department 30, including name, hiredate, salary and commission, if any.

4) Create a view of all managers, including their name, employee number, and department number.

5) Look at your ***MANAGER*** view. Which manager(s) have employee numbers higher than 7650?

6) Which employees from your ***DEPARTMENT 30*** view earn a commission greater than $100/month?

Practice Session Three (continued)

Challenge Exercise

7) Create a view called EMPLOYEES, with the following columns for each employee:

   - Employee name
   - Employee number
   - Employee job

   Do not include clerks in the view.

Practice Session Four

1) Create a report that includes the following:

   - Title - "Employee Records"
     Subtitle - "HiTech Corp"
   - Today's date at the top
   - The page number at the top
   - Footer - "Prepared by a SQL*Plus Student"
   - Employees grouped by department number, and each department separated from the next by two lines
   - Total salaries for each department

2) Look at your report to make sure it is accurate.

Practice Session Four (continued)

Challenge Exercise

3) Prepare a salary report that meets the following format requirements:

   a. The report should include employee name, length of employment in weeks, salary and department number.

   b. Only one department should appear per page.

   c. At the end of each page, present the sum of all salaries for that page as well as the sum of all employment lengths for that page.

   d. At the end of the report, present the total salaries paid to all employees and their total time with the company.

   e. All salaries should be in dollars, and all employment lengths in weeks.

   f. There should be a title at the top of each page.

Practice Session Five

Note: Today's practice sessions involve three new tables:

INVENTION table
NATION table
BORDER table

You can find copies of these tables in the last section.

1) Select the names of all inventors that have more than 2 inventions in the table and order the result by the first letter of their names.

2) How many different inventors are in the table?

3) Find the nations whose population density (population divided by area) is less than the length of the nation's name.

Practice Session Five (continued)

Challenge Exercise

4) Create a report that includes employee name, total compensation and hiredate. Make sure the listing contains:

   a. Names with first letter capitalized

   b. Hiredate in the format MM/DD/YYYY

   c. Total compensation for each employee --- including employees that do not earn any commission

   d. Names in alphabetical order

Practice Session Six

1) Display the name of the department in which SMITH works. Include his name in the display.

2) List all inventors whose names begin with 'B'. Include their countries.

3) Report the total number of inventions from each country of origin found in the ***INVENTION*** table. Be sure to include the name of the country in the result.

4) Select all inventions invented in the same year as any year in which inventor ***'BENZ'*** had an invention.

Challenge Exercises

5) **List the name and size of all nations that have more than 7 bordering nations.**

6) **Find the names and populations of all island nations that have a land area larger than or equal the area of Japan.**

Practice Session Six (continued)

7) Display all the countries which border on other countries, and for each country list its bordering nations by name.

Answers to Practice Session One

*1) Log onto SQL*Plus.*

```
SQLPLUS     <return>
UserID      <return>
Password    <return>
```

2) Retrieve all data from EMP.

```
SELECT * FROM EMP;
```

3) Display the names of all employees who work in department 20. Make sure the column containing employee names is called NAME.

```
SELECT   ENAME NAME
FROM     EMP
WHERE    DEPTNO = 20;
```

4) Display the names of all employees in department 10 and 20, ordering the listing in descending alphabetical order.

```
SELECT      ENAME NAME,
            DEPTNO
FROM        EMP
WHERE       DEPTNO IN (10, 20)
ORDER BY    ENAME DESC;
```

5) Display all department locations that do not start with the letter 'N'.

```
SELECT   LOC
FROM     DEPT
WHERE    LOC NOT LIKE 'N%';
```

Challenge Exercises

6) Do any employees have a salary that is more than their commission? If so, present the listing with salaries in descending order. If two or more employees have the same salary, order them alphabetically by name.

```
SELECT      ENAME EMPLOYEE,
            SAL SALARY,
            COMM COMMISSION
FROM        EMP
WHERE       SAL > COMM
ORDER BY    SAL DESC,
            ENAME;
```

7) Prepare a report containing employee name, commission percent. Ensure that the report does not have any acronyms or abbreviations. Order the listing by employee names. Do not include employees who are not eligible for a commission.

```
SELECT      ENAME EMPLOYEE,
            (COMM / (COMM + SAL)) * 100 INCENTIVE
FROM        EMP
WHERE       COMM IS NOT NULL
ORDER BY    ENAME;
```

Answers to Practice Session Two

1) Add Joe Wilson to your EMP table. Make up information about him to complete your record.

```
INSERT INTO   EMP
VALUES        (7901, 'WILSON', 'ENGINEER', 7839,
              '15-SEP-87', 1500, NULL, 20);
```

2) Promote Joe to Manager.

```
UPDATE    .EMP
SET       JOB = 'MANAGER'
WHERE     ENAME = 'WILSON';
```

3) Joe was fired. Delete his record from the table.

```
DELETE FROM    EMP
WHERE          ENAME = 'WILSON';
```

Challenge Exercises

4) Give all employees in the Chicago office (department 30) a 10% raise, and modify the appropriate table to reflect that.

```
UPDATE    EMP
SET       SAL = 1.1 * SAL
WHERE     DEPTNO = 30;
```

Alternate method:

```
UPDATE    EMP
SET       SAL = 1.1 * SAL
WHERE     DEPTNO =  (SELECT  DEPTNO
                     FROM    DEPT
                     WHERE   LOC = 'CHICAGO');
```

5) Create a Facilities department for Hitech Inc. and assign it number 50. No other information is known yet about this department.

```
INSERT INTO   DEPT (DNAME, DEPTNO)
VALUES        ('FACILITIES', 50);
```

Answers to Practice Session Three

1) *Add a column to the EMP table to store the first name of each employee.*

```
ALTER TABLE   EMP
ADD           (FIRST_NAME CHAR(10));
```

2) *Change EMP so that it can accommodate last names that are 24 characters long.*

```
ALTER TABLE   EMP
MODIFY        (ENAME CHAR (24));
```

3) *Create a view of all employees in department 30, including name, hiredate, salary and commission, if any.*

```
CREATE VIEW   DEPT30
AS SELECT     ENAME, HIREDATE, SAL, COMM
FROM          EMP
WHERE         DEPTNO = 30;
```

4) *Create a view of all managers, including name, employee number and department number.*

```
CREATE VIEW   MANAGER
AS SELECT     ENAME, EMPNO, DEPTNO
FROM          EMP
WHERE         JOB = 'MANAGER';
```

5) *Look at your MANAGER view. Which manager(s) have employee numbers higher than 7650?*

```
SELECT        ENAME, EMPNO
FROM          MANAGER
WHERE         EMPNO > 7650;
```

6) *Which employees from your DEPARTMENT 30 view earn a commission greater than $100/month?*

```
SELECT        ENAME, COMM
FROM          DEPT30
WHERE         COMM > 100;
```

Challenge Exercise

7) Create a view called EMPLOYEE, with the following columns for each employee:

- *Employee name*
- *Employee number*
- *Employee job*

Do not include clerks in the view.

```
CREATE VIEW     EMPLOYEE     ("Employee Name",
                              "Employee Number",
                              "Employee Job")
AS    SELECT    ENAME,
                EMPNO,
                JOB
      FROM      EMP
      WHERE     JOB ! = 'CLERK';
```

Answers to Practice Session Four

1) Create a report that includes the following:

- *Title - "Employee Records"*
  Subtitle - "HiTech Corp"
- *Today's date at the top*
- *The page number at the top*
- *Footer - "Prepared by a SQL*Plus Student"*
- *Employees grouped by department number, and each department separated from the next by two lines*
- *Total salaries for each department*

```
TTITLE 'Employee Records || Hitech Corp'

BTITLE 'Prepared by a SQL*Plus Student'

BREAK ON DEPTNO SKIP 2

COMPUTE SUM OF SAL ON DEPTNO

SELECT      *
FROM        EMP
ORDER BY    DEPTNO;
```

```
column ename heading NAME
column ename format a10
column mgr format 9999
set pagesize 22
```

2) Look at your report to make sure it is accurate.

RUN <return>

Challenge Exercise

3) Prepare a salary report that meets the following format requirements:

a. The report should include employee name, length of employment in weeks, salary and department number.

b. Only one department should appear per page.

c. At the end of each page, present the sum of all salaries for that page as well as the sum of all employment lengths for that page.

d. At the end of the report, present the total salaries paid to all employees and their total time with the company.

e. All salaries should be in dollars, and all employment lengths in weeks.

f. There should be a title at the top of each page.

```
TTITLE 'Service'

BREAK ON DEPTNO PAGE ON REPORT

COMPUTE SUM OF SAL ON DEPTNO

COMPUTE SUM OF SAL ON REPORT

COMPUTE SUM OF SERVICE_LENGTH ON DEPTNO

COMPUTE SUM OF SERVICE_LENGTH ON REPORT

COLUMN SAL FORMAT $99,999.00

COLUMN SERVICE_LENGTH FORMAT 9999

SELECT      DEPTNO,
            ENAME EMPLOYEE,
            (SYSDATE - HIREDATE) / 7 SERVICE_LENGTH,
            SAL
FROM        EMP
ORDER BY    DEPTNO;
```

Answers to Practice Session Five

Note: Today's practice sessions involve three new tables:

INVENTION table
NATION table
BORDER table

You can find copies of these tables in the last section.

1) Select the names of all inventors that have more than 2 inventions in the table and order the result by the first letter of their names.

```
SELECT      INVENTOR, COUNT(*)
FROM        INVENTION
GROUP BY    INVENTOR
HAVING      COUNT(*) > 2
ORDER BY    SUBSTR (INVENTOR,1,1);
```

2) How many different inventors are in the table?

```
SELECT      COUNT (DISTINCT INVENTOR)
FROM        INVENTION;
```

3) Find the nations whose population density (population divided by area) is less than the length of the nation's name.

```
SELECT      NATION, POPULATION/AREA DENSITY
FROM        NATION
WHERE       (POPULATION / AREA) < LENGTH (NATION);
```

Challenge Exercise

4) Create a report that includes employee name, total compensation and hiredate. Make sure the listing contains:

a. Names with first letter capitalized

b. Hiredate in the format MM/DD/YYYY

c. Total compensation for each employee --- including employees that do not earn any commission

d. Names in alphabetical order

```
COL "Hire Date" FORMAT A12

COL "Employee" FORMAT A10

COL "Compensation" FORMAT $99,999.00

SELECT      INITCAP (ENAME) "Employee",
            (SAL + NVL(COMM, 0)) "Compensation",
            TO_CHAR (HIREDATE, 'MM/DD/YYYY') "Hire Date"
FROM        EMP
ORDER BY    ENAME;
```

Answers to Practice Session Six

1) *Display the name of the department in which SMITH works. Include his name in the display.*

```
SELECT      DNAME "Department",
            ENAME "Employee Name"
FROM        EMP,
            DEPT
WHERE       ENAME = 'SMITH'
AND         EMP.DEPTNO = DEPT.DEPTNO;
```

2) *List all inventors whose names begin with 'B'. Please include their countries as well.*

```
SELECT      INVENTOR,
            NATION
FROM        INVENTION,
            NATION
WHERE       INVENTOR LIKE 'B%'
AND         INVENTION.NATION_CODE = NATION.CODE;
```

3) *Report the total number of inventions from each country of origin found in the **INVENTION** table. Be sure to include the name of the country in the result.*

```
SELECT      NATION, COUNT(*)
FROM        NATION, INVENTION
WHERE       NATION_CODE = CODE
GROUP BY    NATION;
```

4) *Select all inventions invented in the same year as any year in which inventor 'BENZ' had an invention.*

```
SELECT      B.INVENTION, B.INVENTOR, B.YEAR
FROM        INVENTION A, INVENTION B
WHERE       A.YEAR = B.YEAR
AND         UPPER (A.INVENTOR) = 'BENZ';
```

Alternate method:

```
SELECT      INVENTION, INVENTOR, YEAR
FROM        INVENTION
WHERE       YEAR IN  (SELECT  YEAR
                      FROM    INVENTION
                      WHERE   UPPER (INVENTOR) = 'BENZ');
```

A

inventor	year
BENZ	1960
	1980
	1960

B

inven	Year
—	1960
—	1980
—	1970

Challenge Exercises

5) List the name and size of all nations that have more than 7 bordering nations.

```
SELECT      NATION, AREA
FROM        NATION
WHERE       CODE IN
            (SELECT     NATION_CODE
            FROM        BORDER
            GROUP BY    NATION_CODE
            HAVING      COUNT(*) > 7);
```

6) Find the names and populations of all island nations that have a land area larger than or equal to the area of Japan.

```
SELECT      NATION, POPULATION
FROM        NATION, BORDER
WHERE       CODE = NATION_CODE(+)
AND         NATION_CODE IS NULL
AND         AREA >=
            (SELECT     AREA
            FROM        NATION
            WHERE       UPPER (NATION) = 'JAPAN');
```

7) Display all the countries which border on other countries, and for each country list its bordering nations by name.

```
BREAK ON        NATION

SELECT          NATION1.NATION,
                NATION2.NATION BORDERING_COUNTRY
FROM            NATION NATION1,
                BORDER,
                NATION NATION2
WHERE           NATION1.CODE = BORDER.NATION_CODE
AND             BORDER.BORDER_CODE = NATION2.CODE
ORDER BY        NATION1.NATION;
```

Tables

EMP Table

```
SQL> describe emp

 Name                          Null?    Type
 ----------------------------- -------- ----
 EMPNO                         NOT NULL NUMBER(4)
 ENAME                                  CHAR(10)
 JOB                                    CHAR(9)
 MGR                                    NUMBER(4)
 HIREDATE                               DATE
 SAL                                    NUMBER(7,2)
 COMM                                   NUMBER(7,2)
 DEPTNO                                 NUMBER(2)

SQL> select * from emp order by empno;

     EMPNO ENAME      JOB       MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ----- --------- ---------- ---------- ----------
      7369 SMITH      CLERK      7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN   7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN   7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER    7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN   7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER    7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER    7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST    7566 07-JUN-87       3000                    20
      7839 KING       PRESIDENT       17-NOV-81       5000                    10
      7844 TURNER     SALESMAN   7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK      7788 11-JUL-87       1100                    20
      7900 JAMES      CLERK      7698 03-DEC-81        950                    30
      7902 FORD       ANALYST    7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK      7782 23-JAN-82       1300                    10

14 records selected.
```

DEPT Table

```
SQL> describe dept
 Name                           Null?    Type
 ------------------------------ -------- ----
 DEPTNO                                  NUMBER(2)
 DNAME                                   CHAR(14)
 LOC                                     CHAR(13)

SQL> select * from dept order by deptno;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

NATION Table

```
SQL> describe nation
 Name                           Null?    Type
 ------------------------------ -------- ----
 CODE                           NOT NULL NUMBER(5)
 NATION                         NOT NULL CHAR(28)
 CAPITAL                                 CHAR(20)
 AREA                                    NUMBER(22)
 POPULATION                              NUMBER(22)

SQL> select * from nation order by code;

 CODE NATION                     CAPITAL                 AREA  POPULATION
----- -------------------------- -------------------- ---------- ----------
    1 Afghanistan                Kabul                   251773    14177000
    2 Albania                    Tirana                   11100     2846000
    3 Algeria                    Algiers                 918497    21300000
    4 Andorra                    Andorra la Vella           188       42000
    5 Angola                     Luanda                  481353     7770000
    6 Antigua and Barbuda        St. John's                 171       80000
    7 Argentina                  Buenos Aires            1065189    29627000
    8 Australia                  Canberra                2966200    15265000
    9 Austria                    Vienna                    32374     7574000
   10 The Bahamas                Nassau                     5380      223000
   11 Bahrain                    Manama                      258      393000
   12 Bangladesh                 Dacca                     55598    96539000
   13 Barbados                   Bridgetown                  166      251000
   14 Belgium                    Brussels                  11779     9865000
   15 Belize                     Belmopan                   8867      154000
   16 Benin                      Porto-Novo                43475     3792000
   17 Bhutan                     Thimphu                   17800     1386000
   18 Bolivia                    Sucre                    424165     5883000
   19 Botswana                   Gaborone                 231804     1001000
   20 Brazil                     Brasilia                3286470   131305000
   21 Brunei                     Bandar Seri Begawan        2226      209000
   22 Bulgaria                   Sofia                     44365     8944000
   23 Burma                      Rangoon                  261288    35480000
   24 Burundi                    Bujumbura                 10759     4561000
   25 Cambodia                   Phnom Penh                69900     5998000
   26 Cameroon                   Yaounde                  179558     9251000
   27 Canada                     Ottawa                  3849670    24882000
   28 Cape Verde                 Praia                      1557      297000
   29 Central African Republic   Bangui                   240324     2512000
   30 Chad                       N'Djamena                495755     4990000
   31 Chile                      Santiago                 292135    11486000
   32 China                      Peking                  3691521  1022054000
   33 China (Taiwan)             Taipei                    13814    18786000
   34 Columbia                   Bogota                   440831    27663000
   35 Comoros                    Moroni                      838      442000
```

CODE	NATION	CAPITAL	AREA	POPULATION
36	Congo	Brazzaville	132046	1694000
37	Costa Rica	San Jose	19653	2624000
38	Cuba	Havana	44218	9889000
39	Cyprus	Nicosia	3572	653000
40	Czechoslovakia	Prague	49365	15420000
41	Denmark	Copenhagen	16633	5115000
42	Djibouti	Djibouti	8996	316000
43	Dominica	Roseau	290	74000
44	Dominican Republic	Santo Domingo	18704	6248000
45	Ecuador	Quito	108624	8811000
46	Egypt	Cairo	385201	45809000
47	El Salvador	San Salvador	8124	4885000
48	Equatorial Guinea	Malabo	10832	268000
49	Ethiopia	Addis Ababa	472400	31265000
50	Fiji	Suva	7056	678000
51	Finland	Helsinki	130119	4850000
52	France	Paris	210040	54652000
53	Gabon	Libreville	102317	921000
54	The Gambia	Banjul	4361	700000
55	East Germany	East Berlin	41825	16724000
56	West Germany	Bonn	96011	61493000
57	Ghana	Accra	92098	13367000
58	Greece	Athens	50962	9898000
59	Grenada	St. George's	133	111000
60	Guatemala	Guatemala City	42042	7714000
61	Guinea	Conakry	94925	5430000
62	Guinea-Bissau	Bissau	13948	827000
63	Guyana	Georgetown	83000	833000
64	Haiti	Port-au-Prince	10714	5690000
65	Honduras	Tegucigalpa	43277	4276000
66	Hungary	Budapest	35919	10691000
67	Iceland	Reykjavik	39769	236000
68	India	New Delhi	1269420	730572000
69	Indonesia	Jakarta	741101	165787000
70	Iran	Teheran	636363	42490000
71	Iraq	Baghdad	168928	14509000
72	Ireland	Dublin	27137	3534000
73	Israel	Jerusalem	8219	3958000
74	Italy	Rome	116303	56833000
75	Ivory Coast	Abidjan	124503	8890000
76	Jamaica	Kingston	4244	2347000
77	Japan	Tokyo	147470	119205000
78	Jordan	Amman	37297	2568000
79	Kenya	Nairobi	224081	18560000
80	Kiribati	Tarawa	266	60000
81	North Korea	Pyongyang	47077	19185000
82	South Korea	Seoul	38211	41386000
83	Kuwait	Kuwait	6532	1652000
84	Laos	Vientiane	91428	3647000
85	Lebanon	Beirut	3950	2598000
86	Lesotho	Maseru	11716	1438000

CODE	NATION	CAPITAL	AREA	POPULATION
87	Liberia	Monrovia	38250	2091000
88	Libya	Tripoli	679536	3498000
89	Liechtenstein	Vaduz	62	26000
90	Luxembourg	Luxembourg	1034	366000
91	Madagascar	Antananarivo	226658	9389000
92	Malawi	Lilongwe	45747	6612000
93	Malaysia	Kuala Lumpur	127316	14995000
94	Maldives	Male	115	168000
95	Mali	Bamako	478841	7393000
96	Malta	Valletta	122	358000
97	Mauritania	Nouakchott	398000	1591000
98	Mauritius	Port Louis	787	1002000
99	Mexico	Mexico City	761604	75702000
100	Monaco	Monaco-Ville	1	28000
101	Mongolia	Ulaanbaatar	604247	1809000
102	Morocco	Rabat	171117	22889000
103	Mozambique	Maputo	303769	13047000
104	Nauru	Yaren	8	8000
105	Nepal	Kathmandu	56136	16169000
106	Netherlands	Amsterdam	16464	14374000
107	New Zealand	Wellington	103883	3203000
108	Nicaragua	Managua	57000	2812000
109	Niger	Niamey	490100	6083000
110	Nigeria	Lagos	356700	85219000
111	Norway	Oslo	125057	4131000
112	Oman	Muscat	115800	978000
113	Pakistan	Islamabad	307374	94140000
114	Panama	Panama	29762	2058000
115	Papua New Guinea	Port Moresby	178704	3259000
116	Paraguay	Asuncion	157047	3526000
117	Peru	Lima	496222	18663000
118	Philippines	Quezon City	115831	54252000
119	Poland	Warsaw	120727	36566000
120	Portugal	Lisbon	36390	1000800
121	Qatar	Doha	4247	267000
122	Romania	Bucharest	91699	22579000
123	Rwanda	Kigali	10169	5644000
124	St. Kitts and Nevis	Basseterre	101	44500
125	Saint Lucia	Castries	238	119000
126	St. Vincent and Grenadines	Kingstown	150	134000
127	San Marino	San Marino	24	22000
128	Sao Tome and Principe	Sao Tome	372	88000
129	Saudi Arabia	Riyadh	830000	10443000
130	Senegal	Dakar	76095	6335000
131	Seychelles	Victoria	171	65000
132	Sierra Leone	Freetown	27699	3705000
133	Singapore	Singapore	238	2501000
134	Solomon Islands	Honiara	10640	254000
135	Somalia	Mogadishu	246300	6248000
136	South Africa	Cape Town	435868	30938000
137	Spain	Madrid	195988	38234000
138	Sri Lanka	Colombo	25332	18300000

CODE	NATION	CAPITAL	AREA	POPULATION
139	Sudan	Khartoum	966757	20539000
140	Suriname	Paramaribo	83037	363000
141	Swaziland	Mbabane	6704	632000
142	Sweden	Stockholm	178896	8331000
143	Switzerland	Bern	15941	6463000
144	Syria	Damascus	71498	9700000
145	Tanzania	Dar-es-Salaam	364886	20524000
146	Thailand	Bangkok	198500	60731000
147	Togo	Lome	21853	2838000
148	Tonga	Nuku'alofa	270	104000
149	Trinidad and Tobago	Port-of-Spain	1870	1149000
150	Tunisia	Tunis	63378	7020000
151	Turkey	Ankara	300948	49155000
152	Tuvalu	Funafuti	10	8000
153	Uganda	Kampala	91104	13819000
154	U.S.S.R.	Moscow	8649490	272500000
155	United Arab Emirates	Abu Dhabi	32000	1374000
156	United Kingdom	London	94222	56009000
157	United States of America	Washington, D.C.	3623420	234249000
158	Upper Volta	Ouagadougou	105869	6569000
159	Uruguay	Montevideo	68037	2916000
160	Vanuatu	Vila	4707	127000
161	Venezuela	Caracas	352143	17993000
162	Vietnam	Hanoi	127207	67612000
163	Western Samoa	Apia	1133	160000
164	North Yemen	Sanaa	77200	5744000
165	South Yemen	Aden	130541	2086000
166	Yugoslavia	Belgrade	98766	22826000
167	Zaire	Kinshasa	905063	31250000
168	Zambia	Lusaka	290586	6346000
169	Zimbabwe	Harare	150873	8077000

169 records selected.

BORDER Table

```
SQL> describe border
 Name                           Null?    Type
 ------------------------------ -------- ----
 NATION_CODE                    NOT NULL NUMBER(4)
 BORDER_CODE                    NOT NULL NUMBER(4)
 NORTH                                   CHAR(1)
 SOUTH                                   CHAR(1)
 EAST                                    CHAR(1)
 WEST                                    CHAR(1)

SQL> select * from border order by nation_code;

NATION_CODE BORDER_CODE N S E W
----------- ----------- - - - -
          1         113   x x
          1          70       x
          1         154 x
          1          32 x   x
          2          68   x
          2         166 x   x
          3         102       x
          3          97   x
          3          95   x
          3         109   x
          3          88     x
          3         150     x
          4         137   x
          4          52 x
          5         168     x
          6         167 x
          7          31       x
          7          18 x
          7         116 x
          7          20 x   x
          7         159 x   x
          8         143       x
          8          89       x
          8          66 x
          8          40 x
          8          86     x
          8         166   x
          8          74   x
         12          68 x   x x
         12          23   x x
         14          52   x   x
         14          90   x x
         14          56     x
```

NATION_CODE	BORDER_CODE	N	S	E	W
14	106	x			
15	99	x			
15	60		x		x
16	147				x
16	158	x			
16	109	x			
16	110			x	
17	68		x		x
17	32	x			
18	31				x
18	117				x
18	7		x		
18	116		x		
18	20	x		x	
19	136	x	x		x
19	169	x		x	
19	168	x			
20	63	x			
20	140	x			
20	161	x			
20	34				x
20	117				x
20	18				x
20	116				x
20	7				x
20	159		x		
21	93		x		
22	122	x			
22	166				x
22	58		x		
22	151		x		
23	12				x
23	68				x
23	32			x	
23	84			x	
23	148			x	
24	123	x			
24	167				x
24	145			x	
25	148	x			x
25	84	x		x	
25	162			x	
26	110	x			x
26	30			x	
26	29			x	
26	36		x		
26	53		x		
26	48		x		
27	157		x		x
29	30	x			
29	26				x

NATION_CODE	BORDER_CODE	N	S	E	W
29	36	x			
29	167	x			
29	139			x	
30	88	x			
30	26				x
30	109				x
30	110				x
30	29		x		
30	139			x	
31	117	x			
31	18	x		x	
31	7			x	
32	101	x			
32	.154	x		x	x
32	1				x
32	113				x
32	68		x		
32	105		x		
32	17		x		
32	23		x		
32	84		x		
32	162		x		
32	81	x			x
34	114	x			x
34	45		x		
34	117		x		
34	20			x	
34	161			x	
36	53				x
36	26				x
36	29	x			
36	167			x	
36	5		x		x
37	108	x			
37	114		x		
40	55	x			x
40	119	x			x
40	56				x
40	9		x		
40	66		x		
40	154			x	
41	66		x		
41	111	x			x
41	142	x		x	
42	49	x			x
42	135		x		
45	34	x			
45	117		x	x	
46	88				x
46	139		x		
46	73			x	
47	60				x
47	65	x			

NATION_CODE	BORDER_CODE	N	S	E	W
48	53		x		
48	26	x		x	
49	139				x
49	79		x		
49	135			x	
49	42			x	
51	111	x			
51	142				x
51	154			x	
52	137		x		
52	74			x	
52	143			x	
52	56			x	
52	90	x			
52	14	x			
53	48	x			
53	28	x			
53	36		x	x	
54	130	x	x	x	
55	56				x
55	40		x		
55	119			x	
56	55			x	
56	41	x			
56	106				x
56	14				x
56	90				x
56	52				x
56	143		x		
56	9		x		
56	40			x	
57	75				x
57	158	x			
57	147			x	
58	2	x			
58	166	x			
58	22	x			
58	151			x	
60	99	x			x
60	47		x		
60	15			x	
60	85			x	
61	62	x			
61	130	x			
61	95	x			
61	75			x	
61	87		x		
62	130	x			
62	61		x	x	
63	161				x
63	20		x		
63	140			x	
65	60				x

NATION_CODE	BORDER_CODE	N	S	E	W
65	47		x		
65	108		x		
66	40	x			
66	9				x
66	166		x		
66	122			x	
66	154			x	
68	113				x
68	32	x			
68	105	x			
68	17	x			
68	23			x	
68	12			x	
69	93	x			
69	115			x	
70	151				x
70	71				x
70	154	x			
70	1			x	
70	113			x	
71	78				x
71	144				x
71	151	x			
71	70			x	
71	83		x		
71	129		x		
72	156	x			
73	85	x			
73	78			x	
73	144			x	
73	46				x
74	62				x
74	9	x			
74	143	x			
74	166			x	
75	61				x
75	87				x
75	158	x			
75	95	x			
75	67			x	
78	73				x
78	120		x		
78	71			x	
78	144	x			
79	153				x
79	145		x		
79	135			x	
79	49	x			
79	139	x			
81	154	x			
81	32	x			
81	82		x		
82	81	x			

NATION_CODE	BORDER_CODE	N	S	E	W
83	71	x			
83	129		x		
84	32	x			
84	23	x			
84	162			x	
84	25		x		
84	146				x
85	144			x	
85	73		x		
86	136	x	x	x	x
87	132				x
87	81	x			
87	75			x	
88	3				x
88	150				x
88	109		x		
88	30		x		
88	46			x	
88	139			x	
89	143				x
89	9			x	
90	14				x
90	52		x		
90	56			x	
92	168				x
92	103		x	x	
92	145	x			
93	146	x			
93	69		x		
95	97				x
95	130				x
95	81		x		
95	75		x		
95	158		x		
95	109			x	
96	3	x			
97	102	x			
97	3			x	
97	95			x	
97	130		x		
99	167	x			
99	15		x		
99	60		x		
100	52	x		x	x
101	154	x			
101	32		x		
102	97		x		
102	3			x	
103	145	x			
103	92				x
103	168				x
103	169				x
103	136		x		

NATION_CODE	BORDER_CODE	N	S	E	W
103	141		x		
105	32	x			
105	68		x		
106	56			x	
106	14		x		
108	65	x			
108	37		x		
109	3	x			
109	88	x			
109	95				x
109	158				x
109	16		x		
109	110		x		
109	30			x	
110	16				x
110	109	x			
110	30			x	
110	26			x	
111	51			x	
111	142			x	
111	154			x	
112	155				x
112	129				x
112	165				x
113	70				x
113	32	x			
113	1	x			
113	88			x	
114	37				x
114	34			x	
115	89				x
116	18	x			
116	7		x		
116	20			x	
117	34	x			
117	45	x			
117	18			x	
117	20			x	
117	31		x		
119	55				x
119	40		x		
119	154			x	
120	137	x		x	
121	129				x
121	155		x		
122	154	x		x	
122	88				x
122	166				x
122	22		x		
123	153	x			
123	167				x
123	24		x		
123	145			x	

NATION_CODE	BORDER_CODE	N	S	E	W
127	74	x	x	x	x
129	83	x			
129	71	x			
129	78	x			
129	164		x		
129	165		x		
129	112		x		
129	155			x	
129	121			x	
130	97	x			
130	95			x	
130	61		x		
130	62		x		
130	34				
132	61	x		x	
132	87		x		
135	42				x
135	49				x
135	79				x
136	19	x			
136	169	x			
136	103			x	
136	141			x	
137	120				x
137	52	x			
138	46	x			
139	88				x
139	30				x
139	29				x
139	167		x		
139	153		x		
139	79		x		
139	49			x	
140	83				x
140	20		x		
140	52			x	
141	136	x	x		x
141	103			x	
142	111				x
142	51			x	
143	52				x
143	74		x		
143	0			x	
143	56	x			
144	85				x
144	73				x
144	78		x		
144	71			x	
144	151	x			
145	79	x			
145	153	x			
145	123				x
145	24				x

NATION_CODE	BORDER_CODE	N	S	E	W
145	167				x
145	168		x		
145	92		x		
145	103		x		
146	23				x
146	84	x			
146	25			x	
146	93		x		
147	67				x
147	158	x			
147	18			x	
150	3				x
150	88			x	
151	22				x
151	58				x
151	154	x			
151	70			x	
151	71		x		
151	144		x		
153	139	x			
153	167				x
153	123		x		
153	145		x		
153	79			x	
154	51				x
154	119				x
154	40				x
154	66				x
154	122				x
154	151		x		
154	70		x		
154	1		x		
154	32		x		
154	101		x		
154	81		x		
155	121	x			
155	129		x		x
155	112			x	
156	72				x
158	52		x	x	
157	27	x			
157	99		x		
158	95	x			x
158	109	x		x	
158	18		x		
158	147		x		
158	57		x		
158	75		x		
159	7				x
159	20	x			
161	34				x
161	20		x		
161	63			x	

NATION_CODE	BORDER_CODE	N	S	E	W
162	32	x			
162	25				x
162	84				x
164	129	x		x	
164	165		x		
165	164				x
165	129	x			
165	112			x	
166	74				x
166	9	x			
166	66	x			
166	22			x	
166	122			x	
166	2		x		
166	58		x		
167	36				x
167	29	x			
167	130	x			
167	153			x	
167	123			x	
167	24			x	
167	145			x	
167	168		x		
167	5		x		
168	167	x			
168	145			x	
168	92			x	
168	103			x	
168	169		x		
168	138		x		
168	5				x
169	168	x			
169	19				x
169	138		x		
169	103			x	

490 records selected.

INVENTION Table

```
SQL> describe invention
 Name                           Null?    Type
 ------------------------------ -------- ----
 INVENTION                      NOT NULL CHAR(30)
 INVENTOR                                CHAR(30)
 YEAR                                    NUMBER(4)
 NATION_CODE                             NUMBER(4)

SQL> select * from invention order by invention;
```

INVENTION	INVENTOR	YEAR	NATION_CODE
Adding Machine	Pascal	1642	52
Addressograph	Duncan	1892	157
Aerosol Spray	Goodhue	1941	157
Air Brake	Westinghouse	1868	157
Air Conditioning	Carrier	1911	157
Air Pump	Guericke	1650	56
Airplane Jet Engine	Ohain	1939	56
Airplane with Motor	Wright bros	1903	157
Airship, Rigid Dirigible	Zeppelin	1900	56
Antiseptic Surgery	Lister	1867	156
Aspirin	Dresser	1889	56
Atomic Theory	Dalton	1803	156
Automobile, Electric	Morrison	1892	157
Automoblie, Gasoline	Daimler	1887	66
Automobile, Steam	Roper	1889	157
Bacteria (described)	Leeuwenhoek	1676	106
Balloon	Montgolfier	1783	52
Barometer	Torricelli	1643	74
Bicycle	Starley	1884	156
Bifocal Lens	Franklin	1780	157
Blood, Circulation	Harvey	1628	156
Bottle Machine	Owens	1903	157
Braille Printing	Braille	1829	52
Burner, Gas	Bunsen	1855	66
Calculating Machine	Babbage	1823	156
Calculus	Newton	1670	156
Canning (food)	Appert	1804	52
Camera, Kodak	Eastman, Walker	1888	157
Camera, Polaroid Land	Land	1948	157
Carburetor, Gasoline	Daimler	1876	66
Cash Register	Ritty	1879	157
Cathode Ray Tube	Crookes	1878	156
Cellophane	Brandenberger	1911	143
Chronometer	Harrison	1735	156
Circuit Breaker	Hillard	1925	157

INVENTION	INVENTOR	YEAR	NATION_CODE
Clock, Pendulum	Huygens	1657	106
Combustion Explained	Lavoisier	1777	52
Computer, Automatic Sequence	Aiken et al	1939	157
Condenser Microphone	Wente	1920	157
Conditioned Reflex	Pavlov	1914	154
Cortisone	Kendall	1936	157
Cosmic Rays	Gockel	1910	143
Cotton Gin	Whitney	1793	157
Cream Separator	DeLaval	1880	142
Cultivator, Disc	Mellon	1878	157
Cyclotron	Lawrence	1930	157
DDT	Zeidler	1874	56
DNA (structure)	Crick, Watson	1951	156
Diesel Engine	Diesel	1895	56
Dynamite	Nobel	1866	142
Electric Battery	Volta	1800	74
Electric Resistance (law)	Ohm	1827	56
Electric Waves	Hertz	1888	56
Electroencephalograph	Berger	1929	56
Electrolysis	Faraday	1852	156
Electromagnet	Sturgeon	1824	156
Electromagnetism	Oersted	1819	41
Electron	Thompson, J.	1897	156
Electroplating	Brugnatelli	1805	74
Electrostatic Generator	Van de Graaff	1929	157
Elevator Brake	Otis	1852	157
Engine, Automobile	Benz	1879	56
Engine, Coal-Gas 4-Cycle	Otto	1877	56
Engine, Electric Ignition	Benz	1880	56
Engine, Gasoline	Daimler	1886	56
Engine, Steam, Piston	Newcomen	1705	156
Engine, Steam, Piston	Watt	1769	156
Evolution, Natural Selection	Darwin	1858	156
Falling Bodies, Law	Galileo	1590	74
Gas Lighting	Murdoch	1792	156
Gasoline, High Octane	Ipatieff	1930	154
Geiger Counter	Geiger	1913	56
Geometry, Analytic	Descartes	1619	62
Glass, Laminated Safety	Benedictus	1909	62
Glider	Cayley	1853	156
Gravitation, Law	Newton	1687	156
Gyroscope	Foucault	1852	62
Harvester-Thresher	Matteson	1888	157
Helicopter	Sikorsky	1939	157
Holograph	Gabor	1948	156
Human Heart Transplant	Barnard	1967	136
Ice-Making Machine	Gorrie	1851	157
Induction, Electric	Henry	1830	157
Intelligence Testing	Binet, Simon	1905	62
Kaleidoscope	Brewster	1817	156
Lamp, Arc	Brush	1879	157
Lamp, Incandescent	Edison	1879	157
Lamp, Mercury Vapor	Hewitt	1912	157

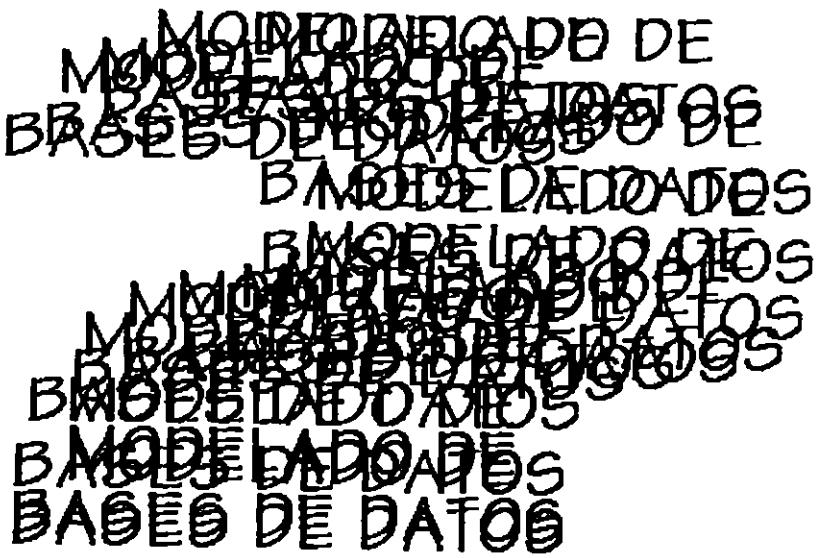
INVENTION	INVENTOR	YEAR	NATION_CODE
Lamp, Neon	Claude	1915	52
Laser	Townes, Schawlow	1958	157
Lathe, Turret	Fitch	1845	157
Lawn Mower	Hills	1868	157
Light, Velocity	Roemer	1675	41
Lightning Rod	Franklin	1752	157
Lock, Cylinder	Yale	1865	157
Locomotive Practical	Stephenson	1829	156
Loom, Power	Cartwright	1785	156
Machine Gun	Gatling	1861	157
Mason Jar	Mason, J.	1858	157
Match, Friction	John Walker	1827	156
Mercator Projection (map)	Mercator (Kremer)	1568	14
Metronome	Malzel	1816	9
Micrometer	Gascoigne	1638	156
Microphone	Berliner	1877	157
Microscope, Compound	Janssen	1590	108
Molecular Hypothesis	Avogadro	1811	74
Motion, Laws Of	Newton	1687	156
Motorcycle	Daimler	1885	56
Movie, Talking	Warner Bros	1927	157
Neoprene	Carothers	1930	157
Neutron	Chadwick	1932	156
Nitroglycerin	Sobrero	1846	74
Parachute	Blanchard	1785	52
Oracle OBMS	Ellison	1979	157
Oxygen	Priestley	1774	156
Pen, Ballpoint	Loud	1888	157
Pen, Fountain	Waterman	1884	157
Penicillin	Fleming	1929	156
Periodic Law & Elements Table	Mendeleyev	1869	154
Pendulum	Galileo	1581	74
Phonograph	Edison	1877	157
Photo, Color	Ives	1892	157
Photo Film, Transparent	Eastman, Goodwin	1878	157
Photoelectric Cell	Elster	1895	56
Photographic Paper	Baekeland	1898	157
Photography	Talbot	1836	156
Piano	Cristofori	1709	7
Piano, Player	Fourneaux	1863	52
Pin, Safety	Hunt	1849	157
Pistol (revolver)	Colt	1835	157
Planetary Motion, Laws	Kepler	1609	56
Plow, Cast Iron	Newbold	1797	157
Plow, Disc	Hardy	1896	157
Positron	Anderson	1932	157
Propeller, Screw	Stevens	1804	157
Proton	Rutherford	1919	107
Psychoanalysis	Freud	1900	9
Punch Card Accounting	Hollerith	1884	157
Quasars	Matthews, Sandage	1963	157
Radar	Taylor, Young	1922	157
Radio Tube-Diode	Fleming	1905	156

INVENTION	INVENTOR	YEAR	NATION_CODE
Radio, Signals	Marconi	1895	74
Radioactivity	Becquerel	1896	52
Razor, Electric	Schick	1931	157
Razor, Safety	Gillette	1895	157
Reaper	McCormick	1834	157
Record, Cylinder	Bell, Tainter	1887	157
Record, Disc	Berliner	1887	157
Record, Long Playing	Goldmark	1948	157
Record, Wax Cylinder	Edison	1888	157
Relativity Theory	Einstein	1905	66
Rocket Engine	Goddard	1929	157
Rubber, Vulcanized	Goodyear	1839	157
Sewing Machine	Howe	1846	157
Shrapnel Shell	Shrapnel	1784	156
Silicon	Berzelius	1823	142
Sleeping-Car	Pullman	1858	157
Slide Rule	Oughtred	1620	156
Spinning Jenny	Hargreaves	1767	156
Steamboat, Practical	Symington	1802	156
Steam Turbine	Parsons	1884	156
Steel	Bessemer	1856	156
Steel, Stainless	Brearley	1916	156
Stethoscope	Laennec	1819	157
Stock Ticker	Edison	1870	157
Stove, Electric	Hadaway	1896	157
Submarine	Holland	1891	157
Tank, Military	Swinton	1914	156
Tape recorder, Magnetic	Poulsen	1899	41
Telegraph, Magnetic	Morse	1837	157
Telegraph, Wireless	Marconi	1896	74
Telephone	Bell	1876	157
Telescope	Galileo	1609	74
Telescope, Astronomical	Kepler	1611	66
Teletype	Morkrum, Kleinschmidt	1928	157
Television, Electronic	Farnsworth	1927	157
Thermometer	Galileo	1593	74
Thermometer, Mercury	Fahrenheit	1714	66
Tire, Pneumatic	Dunlop	1888	166
Toaster, Automatic	Strite	1918	157
Transistor	Shockley,Brattain,Bardeen	1947	157
Turbine, Gas	Curtis, C.G.	1899	157
Turbine, Hydraulic	Francis	1849	157
Turbine, Steam	Curtis, C.G.	1896	157
Type, Movable	Gutenberg	1450	66
Typewriter	Soule, Glidden	1868	157
Vaccine, Polio	Salk	1953	157
Vaccine, Polio, Oral	Sabin	1955	157
Vaccine, Rabies	Pasteur	1885	52
Vaccine, Smallpox	Jenner	1796	156
Vaccine, Typhus	Nicolle	1909	52
Vacuum Cleaner, Electric	Spangler	1907	157
Washer, Electric	Hurley Co.	1907	157
Welding, Electric	Thomson	1877	157

INVENTION	INVENTOR	YEAR	NATION_CODE
Wind Tunnel	Munk	1923	157
Wire, Barbed	Glidden	1874	157
Xerography	Carlson	1938	157
X-Ray	Roentgen	1895	56
Zipper	Judson	1891	157

100 records selected.

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA



MODELADO DE BASES DE DATOS

Material Elaborado por:
Jorge Barba Atilano
Gloria Morales Sánchez

BASES DE DATOS

OBJETIVO GENERAL:

El participante aprenderá a modelar situaciones con el enfoque relacional para el desarrollo de sistemas.

El asistente aplicará instrucciones para la obtención y manipulación de datos mediante un Manejador de Bases de Datos relacional.

Reseña histórica

Las primeras bases de datos estuvieron escritas sobre papel, a partir de los 60's surge el concepto de Base de Datos en computadoras y el primer lenguaje que maneja éstas fue COBOL (COmmon Business Oriented Language).

En 1970 se publica la definición del modelo relacional desarrollado por el Dr. Codd, basado en la simplicidad matematica; la simplicidad matemática es ahora conocida como ALGEBRA BOOLEANA.

En 1978 surge ORACLE, primera implementación real del modelo relacional, surge también SQL.

En los 80's surgen muchos manejadores relacionales.

En 1986 el Dr. Date define las bases de datos distribuidas.

En 1993 ORACLE implementa las bases de datos distribuidas.

Base de datos.

Una BD (Base de Datos) es una colección integrada de datos, conformada por objetos tales como:

Tablas
Indices
Secuencias
Sinónimos
Vistas
Usuarios, etc.

SISTEMA DE ADMINISTRACION DE BASE DE DATOS.

Un sistema de manejo de base de datos (en inglés, DBMS, database management system), consiste en un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a esos datos. El conjunto de datos se conoce comúnmente como base de datos. Esta contiene información acerca de una empresa determinada. El objetivo primordial de un DBMS es crear un ambiente en que sea posible guardar y recuperar información de la base de datos en forma conveniente y eficiente.

Los sistemas de base de datos se diseñan para manejar grandes cantidades de información. El manejo de los datos incluye tanto la definición de las estructuras para el almacenamiento de la información como los mecanismos para el manejo de la información. Además, el sistema de base de datos debe cuidar la seguridad de la información almacenada en la base de datos, tanto contra las caídas del sistema como contra los intentos de acceso no autorizado.

Debido a la importancia que tiene la información en todas las organizaciones, la base de datos es un recurso valioso.

SISTEMA DE PROCESAMIENTO DE ARCHIVOS.

Considérese la parte de una empresa bancaria que conserva la información acerca de todos los clientes y las cuentas de ahorros que tiene el banco. Los registros referentes a las cuentas de ahorros y los clientes se mantienen en archivos permanentes del sistema. Además de estos archivos, el sistema cuenta con varios programas de aplicaciones que permiten manejar los archivos, entre ellos:

- Un programa para hacer cargos o abonos a una cuenta.
- Un programa para agregar una nueva cuenta.
- Un programa para obtener el saldo de una cuenta.
- Un programa para generar estados de cuenta mensuales.

Estos programas de aplicaciones los escribieron programadores del sistema en respuesta a las necesidades de la organización bancaria. Cada vez que se necesita, se agregan nuevos programas de aplicaciones al sistema. Por ejemplo, supóngase que una nueva ley del gobierno permite a la institución de ahorro ofrecer cuentas de cheques. Como resultado, se crean nuevos archivos permanentes que contienen toda la información acerca de las cuentas de cheques que maneja el banco, para lo cual tal vez sea preciso escribir nuevos programas de aplicaciones. Así, con el correr del tiempo, se agregan más archivos y más programas de aplicaciones al sistema. Puesto que estos archivos y programas se han creado en un período largo y, probablemente, por distintos programadores, es de esperar que los archivos tengan formatos diferentes y que los programas esten escritos en varios lenguajes de programación. El ambiente que se acaba de describir es un sistema de procesamiento de archivos característico, apoyado por un sistema operativo convencional. Los registros permanentes se guardan en diversos archivos, y se escriben en varios programas de aplicaciones para sacar registros y agregarlos a los archivos apropiados. Este sistema tiene ciertas desventajas importantes:

REDUNDANCIA E INCONSISTENCIA DE LOS DATOS. Puesto que los archivos y los programas de aplicaciones fueron creados por distintos programadores en un período largo, es posible que un mismo dato este repetido en varios sitios (archivos). Por ejemplo, la dirección y el número telefónico de un cliente determinado pueden aparecer en un archivo que consta de registros de cuentas de ahorros, y en un archivo que se integra con registros de cuentas de cheques. Esta redundancia aumenta los costos de almacenamiento y acceso, además de incrementar la posibilidad de que exista inconsistencia en la información, es decir, que las distintas copias de la misma información no concuerdan entre sí. Por ejemplo, si un cliente cambia de domicilio, y este cambio se refleja únicamente en los registros de cuentas de ahorros, el resultado será inconsistencia de la información.

DIFICULTAD PARA TENER ACCESO A LOS DATOS. Supóngase que uno de los gerentes del banco necesita averiguar los nombres de todos los clientes que viven en cierta parte de la ciudad. El gerente llama al departamento de procesamiento de datos y pide que generen la lista correspondiente. Como esta es una solicitud fuera de lo común, que no estaba prevista cuando se diseñó el sistema original, no existe un programa de aplicaciones para generar semejante lista. Lo que si existe es un programa de aplicaciones que genera una lista de todos los cuentahabientes. El oficial del banco tiene, pues, dos opciones. O bien solicita la lista de todos los clientes y le pide a una de sus secretarias que extraiga manualmente la información requerida, o le solicita al departamento de procesamiento de datos que ponga a uno de los programadores del sistema a escribir el programa de aplicaciones correspondiente. Es obvio que ninguna de las dos opciones es satisfactoria. Supóngase que si se escribe el programa requerido, y que varios días después el mismo gerente necesita eliminar de la lista a todos los clientes cuyo saldo sea inferior a $10 000. Como se suponía, no existe un programa que genere esa lista; una vez más se esta frente a las dos opciones mencionadas anteriormente, ninguna de las cuales es satisfactoria.

Lo que se trata de probar aquí es que este ambiente no permite recuperar la información requerida en forma eficiente. Deben desarrollarse sistemas de recuperación de información de aplicación general.

AISLAMIENTO DE LOS DATOS. Puesto que los datos estan repartidos en varios archivos, y estos pueden tener diferentes formatos, es difícil escribir nuevos programas de aplicaciones para obtener los datos apropiados.

PROBLEMAS DE SEGURIDAD. No es recomendable que todos los usuarios del sistema de base de datos puedan tener acceso a toda la información. Por ejemplo, en un sistema bancario, una persona que prepare los cheques de nómina sólo debe poder ver la parte de la base de datos que contenga información acerca de los empleados del banco. No puede consultar la información correspondiente a las cuentas de los clientes. En forma similar, los cajeros solo pueden tener acceso a la información correspondiente a las cuentas; no pueden consultar la información referente a los salarios de los empleados. Puesto que los programas de aplicaciones se agregan al sistema en una forma precisa, es difícil implantar estas limitantes de seguridad.

PROBLEMAS DE INTEGRIDAD. Los valores de los datos que se guardan en la base de datos deben satisfacer ciertos tipos de limitantes de consistencia. Por ejemplo, el saldo de una cuenta bancaria no debe bajar nunca de un límite fijado (p.e. N$25). El sistema debe obligar al cumplimiento de estas limitantes. Esto puede hacerse agregando el código apropiado a los distintos programas de aplicaciones. Sin embargo, cuando se añaden nuevas limitantes, es difícil cambiar los programas para asegurar su cumplimiento. El problema se complica aún más cuando las limitantas implican varios elementos de información de distintos archivos.

Estos problemas, entre otros, han fomentado el desarrollo de los sistemas de manejo de bases de datos.

MANEJADOR DE BASE DE DATOS.

Un sistema manejador de base de datos (DataBase Management System -DBMS-) es un programa de software que:

- Almacena Datos
- Permite obtener datos
- Modifica datos
- Mantiene la consistencia de los datos
- Resuelve problemas de concurrencia
- Controla el acceso a los datos.
- Permite respaldo y recuperación de la información.

MANEJADOR DE BASES DE DATOS VS. MANEJADOR DE ARCHIVOS.

MANEJADOR DE BASES DE DATOS.

VENTAJAS

Redundancia controlada.
Seguridad a nivel de campos.
Integridad.
Independencia Datos Programas
Recuperación de información
Control de concurrencia.
Respaldos en línea.

DESVENTAJAS

Alto Costo
Bajo Rendimiento

MANEJADOR DE ARCHIVOS.

VENTAJAS
Redundancia
Incongruencia
Programas dependientes de los datos
Seguridad solo a nivel de archivo

DESVENTAJAS
Menor costo
Alto rendimiento (Alta velocidad)

GENERACIONES DE LENGUAJES

PRIMERA GENERACION

0110 1000 1111 0011

SEGUNDA GENERACION

```
LOAD A,10
MOV  A,B
SUM  A,B,C
```

TERCERA GENERACION

```
PROGRAM ALUMNOS
      NOMBRE CHAR(30)
      OPEN (1,'ALUMNOS')
      DO WHILE(.TRUE.)
            READ(1,10,END=100)NOMBRE
10 FORMAT(A,30)
            WRITE(*,*)NOMBRE
      ENDDO
100 CONTINUE
END
```

CUARTA GENERACION

```
SQL> SELECT * FROM ALUMNOS;
```

LAS TRES GENERACIONES DE DBMS's

Jerárquica, de Red y Relacional.

MANEJADOR DE BASES DE DATOS RELACIONALES

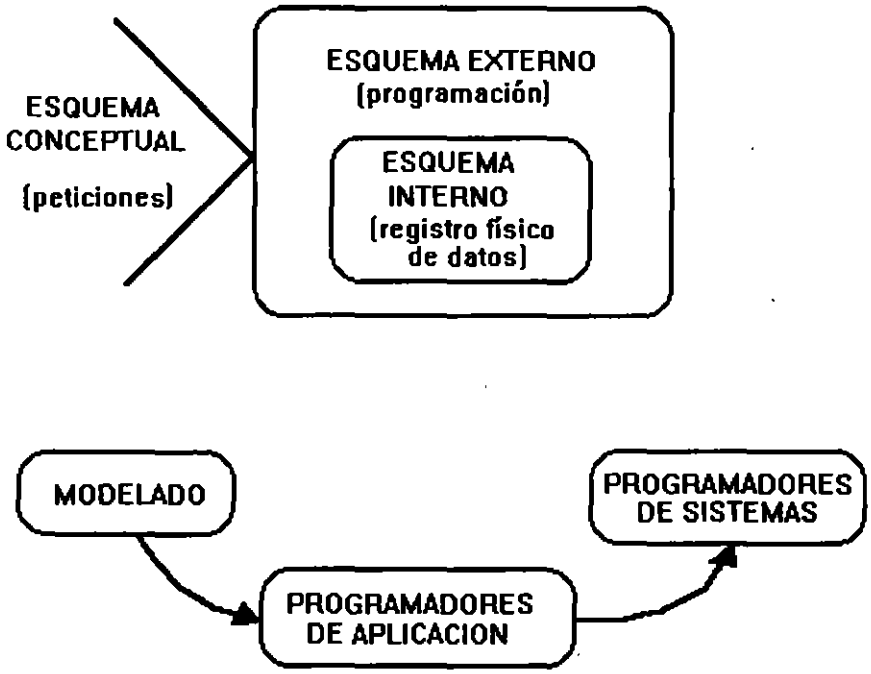
Características:

- Representación de los datos por medio de tablas relacionadas.
- UTILIZA LENGUAJE DE CUARTA GENERACION.
- Flexibilidad:
  la modificación de los datos
  y la estructura de la Base de Datos resulta muy sencillo.
- Contiene un diccionario de datos.

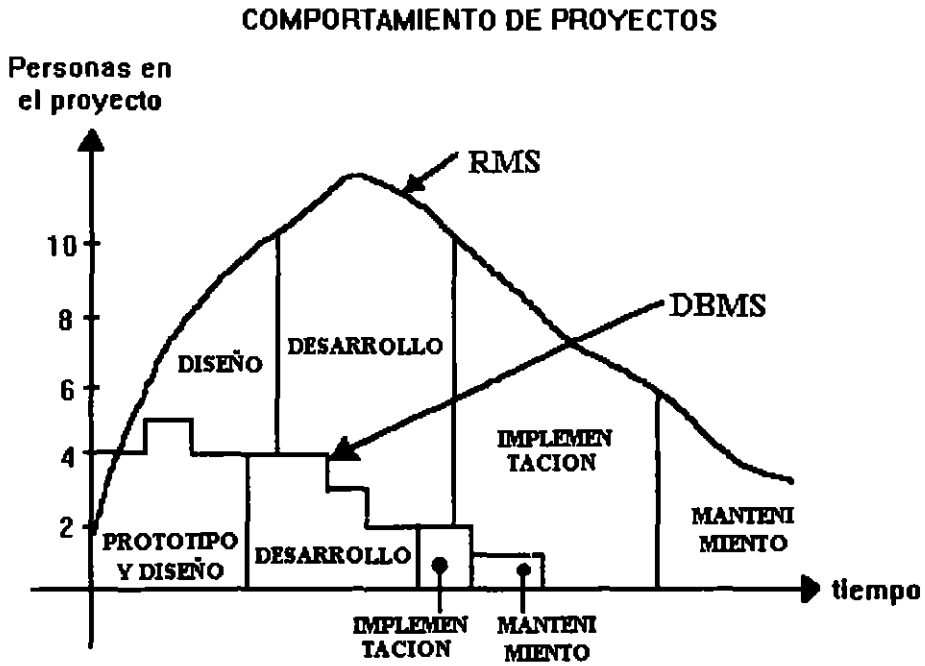
EL ENFOQUE RELACIONAL.

- Un sistema de información de bases de datos relacional se organiza en forma de tablas.
- Las tablas se organizan en renglones y columnas.
- Cada renglón se denomina registro y es información referente a una instancia.
- Cada columna se denomina campo y es información de un solo tipo para todas las instancias.
- De esta forma tu puedes mediante la observación de las tablas entender y utilizar la información que te proporcionan.

ARQUITECTURA DE TRES NIVELES



COMPORTAMIENTO DE PROYECTOS



MODELADO

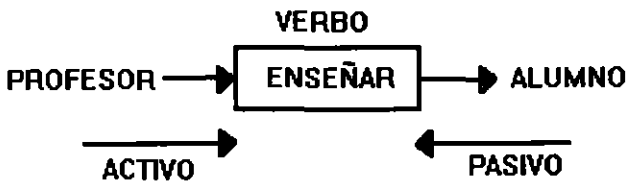
Definiciones:

Situación: Es un conjunto de circunstancias bien definidas que pueden ser descritas utilizando un lenguaje natural suficientemente completo.

Un lenguaje natural sufucientemente completo incluye al menos los siguientes tres constructores gramaticales:

1. Sujetos. Es el nombre de una persona, de un animal, una planta, un lugar, una cosa, una sustancia o una idea. Un **sujeto propio** es el nombre de una ocurrencia particular o **instancia** de un sujeto. Un **pronombre** es una palabra utilizada como sustituto de un sujeto que hace referencia a un sujeto que ya ha sido nombrado o que se sobreentiende de acuerdo al contexto en el cual se utiliza. Los sujetos pueden ser tangibles o intangibles y es imposible describir alguna **situación** sin el uso al menos de un sujeto.

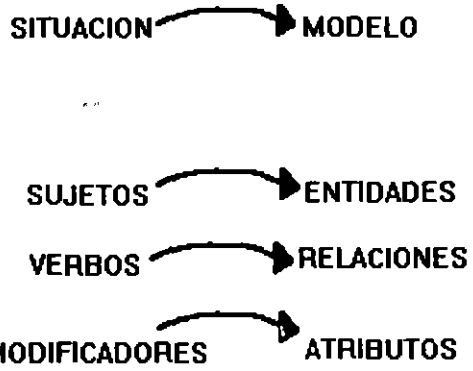
2. Verbos. Es una palabra que describe un modo de ser, una asociación, una acción o un evento. Los verbos describen el estado de los sujetos, y relacionan los sujetos dentro de una situación. Los verbos pueden ser activos o pasivos.



3. Modificadores. Es una palabra que califica a un sujeto o a un verbo de acuerdo a sus características, cantidad, extensión, etc. Los modificadores de los **sujetos** se llaman **adjetivos** y los modificadores de los **verbos** se llaman **adverbios**.

Las situaciones que se describen sin el uso de modificadores son demasiado triviales para ser de interés, sin embargo existen.

MODELADO



MODELO: Representación de una situación mediante la abstracción de un hecho real.

MODELO DE DATOS RELACIONAL

Conexión finita de tablas (de dos dimensiones formadas por columnas y renglones) que representan una situación.

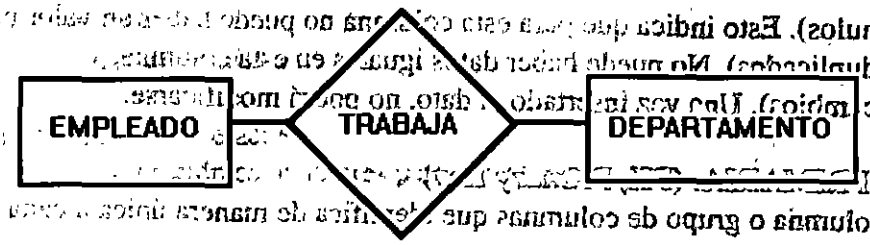
DEPARTAMENTO

CLAVE	NOMBRE
VE	VENTAS
CO	COBRANZAS
IN	INVESTIGACION
ME	MERCADOTECNIA
ES	ESTADISTICA

EMPLEADO

NUMERO	NOMBRE	SALARIO	DEPTO	PUESTO
1	JUAN	2000	VE	VENDEDOR
2	RICARDO	3000	ES	PROGRAMADOR
3	ROSA	2700	IN	INVESTIGADORA
4	LOURDES	2000	VE	VENDEDOR

A cada sujeto le corresponde una tabla.



CARACTERISTICAS DEL MODELO RELACIONAL

SIMPLICIDAD. Las tablas son de una forma familiar y explicables por sí mismas para representar los datos. La mayoría de la gente ha utilizado datos en forma de tabla, no se requiere un entrenamiento especial para entender y utilizar los datos que se representan en las tablas. En pocas palabras son amigables a los usuarios.

PRECISION. Las tablas correctamente diseñadas mantienen un rigor matemático, dicen lo que significan y significan lo que dicen, pueden ser implementadas y procesadas por una variedad de configuraciones de software y hardware. En pocas palabras son amigables a la computadora.

FLEXIBILIDAD. Las tablas no solo muestran la estructura de los datos sino que pueden mostrar los datos también. Esto nos permite manejar el modelo antes de implementarlo. En otras palabras las tablas son apropiadas no solo para modelar datos, sino para procesarlos también.

CARACTERISTICAS DE LAS TABLAS

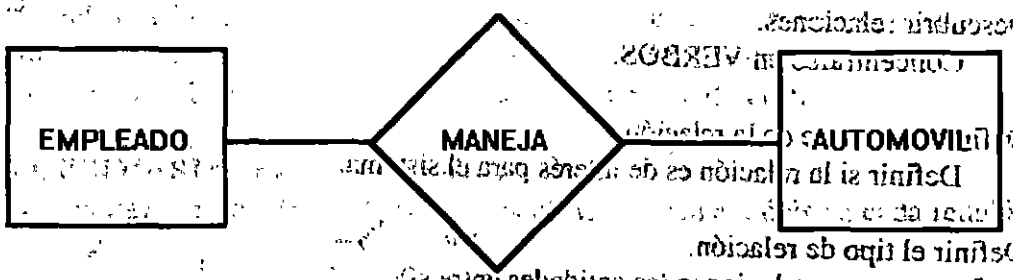
Definición. Una tabla es una representación de datos en dos dimensiones, con una, o más columnas y cero o más renglones.

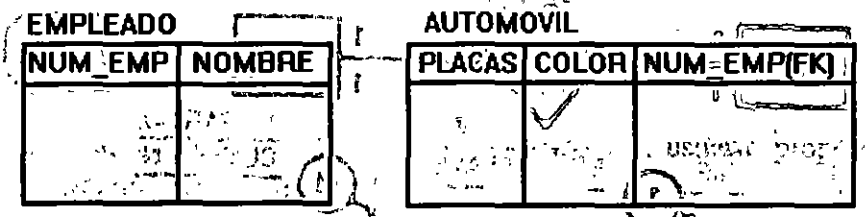
Reglas:

1. El nombre de cada tabla es único para el modelo.
2. El nombre de cada columna es único para la tabla.
3. Cada renglón es único.
4. No es posible descomponer columnas.
5. El orden de las columnas es arbitrario.
6. El orden de los renglones es arbitrario.

NULO. Valor que indica ausencia de la información.
Los nulos no ocupan almacenamiento.

Relación 1:1





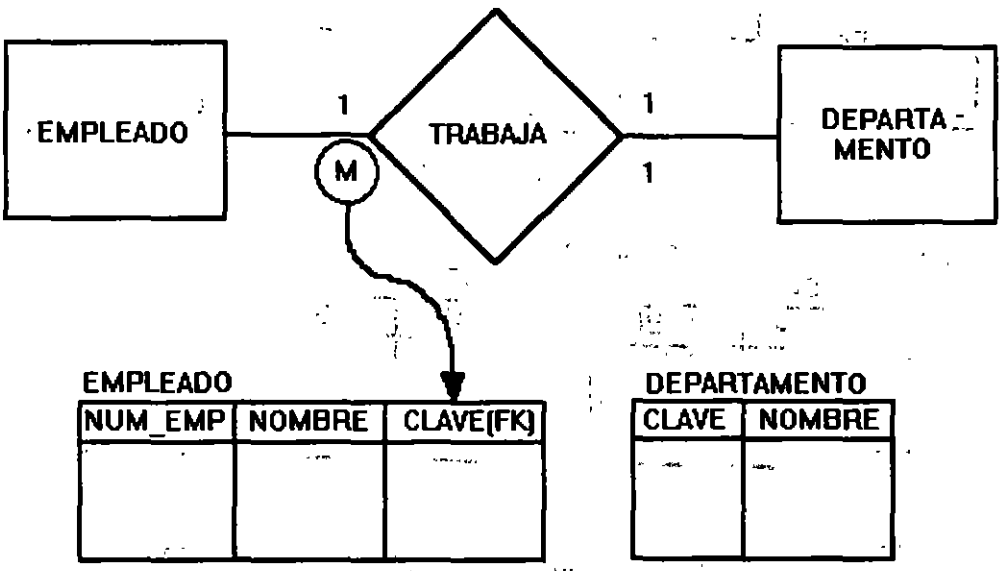
Se puede colocar la llave foránea en cualquiera de las dos tablas pero siempre hay una tabla en donde es mejor.

En este caso, se colocaría en la tabla de automóvil para tener los datos concentrados porque en la tabla de empleado habría varios nulos en caso de que hubiera empleados sin automóvil sin embargo un automóvil siempre estará relacionado con un empleado.

El espacio de almacenamiento es el mismo porque los nulos no ocupan espacio.

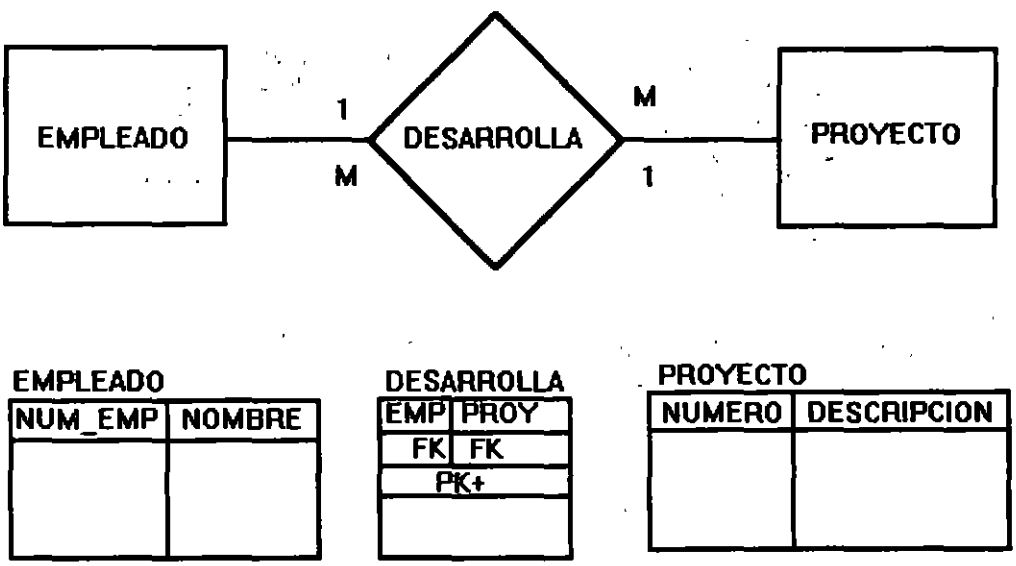
- Relación 1-M y M-1

Esta es un tipo de relación no simétrica, solo puedo colocar la llave foránea FK en la tabla en donde la relación es M.



- Relación M-M

En este tipo de relación se concatenan las llaves foráneas y se crea una nueva tabla, esta tabla recibe el nombre de tabla COMPUESTA.



No importa el orden de las FK, de ahí se observa la simetría de la relación Muchos a Muchos. La tabla DESARROLLA se llama **TABLA COMPUESTA.**

MODELADO DE ATRIBUTOS.

1. Describir atributos

   Concentrarse en adjetivos o adverbios.

2. Definir el alcance del atributo

   Decidir si el atributo es de interés al sistema.

3. Determinar una llave primaria para el atributo

   Determinar cual es la mejor opción para colocar el atributo en una tabla (eliminar duplicidad colocando los atributos de manera correcta.

4. Documentar el atributo en alguna tabla

METODOS PARA COLOCAR ATRIBUTOS.

1. Método gramatical (por experiencia)
2. Normalización

NORMALIZACION

Primera forma normal. (Dependencia entre la PK y un atributo)

Dada una tabla **T**, con una llave primaria **P** y un atributo **A**, se dice que **T** está en primera forma normal, si y sólo si el valor del atributo **A** en cualquier renglón depende del valor de la llave primaria **P** en ese renglón.

T

P	A
1	4
2	4
3	4
4	4

Si todos se repiten
No esté en primera
forma Normal

Puede repetirse
pero no todos
iguales

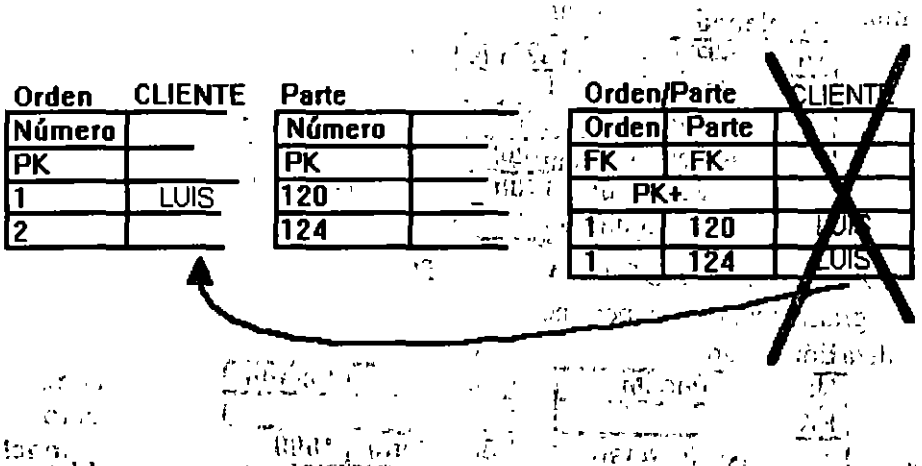
T_2

P	A
1	4
2	4
3	4
4	15

Por ejemplo, un descuento general para los trabajadores, sería un mismo dato para todos y no se incluiría como atributo.

Segunda forma normal. (Dependencia entre un atributo y una FK en una tabla compuesta)

Dada una tabla T en primera forma normal, con una llave primaria P en varias columnas con componenetes P1 y P2 y un atributo A se dice que T está en segunda forma normal, si y sólo si el valor del atributo A en cualquier renglón depende de los dos valores P1 y P2 en ese renglón, las tablas con llaves primarias en una solo columna siempre están en segunda forma normal.



Se aplica en tablas compuestas.

Tercera forma normal.

Dada una tabla T en segunda forma normal, con una llave primaria P y dos atributos A1 y A2, se dice que T está en tercera forma normal, si y solo si el valor del atributo A1 en cualquier renglón no depende del valor del atributo A2, a menos que A2 esté marcado como ND, y el valor del atributo A2 no dependa del valor del atributo A1 en ningún renglón, a menos que A1 esté marcado como ND.

Empleado

Número	Puesto	Salario
PK		
1	Secretaria	1000
2	Secretaria	1000
3	Vendedor	1400
4	Vendedor	1400

No está en tercera forma normal

Empleado

Número	Puesto
PK	
1	Secretaria
2	Secretaria
3	Vendedor
4	Vendedor

Puesto

Puesto	Salario
PK	
Secretaria	1000
Vendedor	1400

Está en tercera forma normal

Al terminar nuestro modelo en tercera forma normal encontramos algunos datos duplicados por lo que a continuación se realiza un proceso de desnormalización.