Database and RDBMS Concepts

Pradeep LN

What is a
Data Base Management System?

Introduction to Database Management Systems

- The most traditional/common way of managing data is to store them in files.
- File processing system
  - Files can be
    - Sequential
    - Indexed
    - Relative
- In this kind of scenario, the files are managed directly by the operating system services
- The primary focus here is to manage the files effectively rather than the contents of them
- Though this mechanism is very good, it has many draw backs

Types of files

- Data File
- Others
  - Source File
  - Object File
  - Library File
  - Executable File
  - Temporary Files
  - List File

Some major drawbacks of File Processing System

- **File processing system manages files rather than the data in them.**
- **File Processing System does not provide effective security**
  - **Either the entire file is secured or it is not**
  - **Some part of the file cannot be secured**
    - **record or column level security is not available**
- **Any program which works with files has to know the physical as well as the logical structure of the file**
  - **Any change in either physical or logical structure of the file, makes it necessary for the program to be rewritten**
    - **Physical Data Dependence**
    - **Logical Data Dependence**

Introduction to DBMS

- Data Base Management System
  - A DBMS is a special set of software which is used to manage data (information)
- Following are some of the important functions of a DBMS
  - Data is Accurate (Accuracy)
  - Data is provided within a timeframe (Timeliness)
  - Only required data is provided (Relevancy)

Why Database ?

- A database system provides a central control of its data
- This is very different from traditional file processing systems where each application / department has its own set of data (Data redundancy)

Advantages of Database

- A database is a collection of data
- A file is a collection of bits and pieces stored together as a single entity.
  - A database system internally relies on the file processing system to manage its data
  - Externally to the user, he feels that he is storing data, rather than a file

Advantages of Database

- Redundancy can be reduced
- Inconsistency can be avoided
- Sharing of Data
- Standards can be enforced
- Security restrictions can be applied
- Integrity of data can be maintained
- Conflicting requirements can be balanced

Characteristics of DBMS

- Data independence
- Speedy handling of spontaneous information requests
- Non-Redundancy
- Versatility in representing relationships between data items
- Security protection
- Real Time accessibility

Relational Model

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris .
S3	Blake	30	Paris .

P

P#	Pname	Color	Weight	City .
P1	Nut	Red	12	London .
P2	Screw	Red	17	Paris .
P3	Bolt	Blue	17	Rome .
P4	Screw	Yellow	14	London .

SP

S#	P#	Qty
S1	P1	200
S1	P2	100
S1	P3	300
S1	P4	230
S2	P2	100

Relational Model

- In relational model the data is simply represented in the form of tables
- If you compare, these three tables closely resemble sequential files
- Compared to the other two models, relational is simple to understand
- There are no links or pointers which connect different tables
- The model is called relational because it uses relational algebra to represent and manage information

More On Relational Model

- Each row is called as a tuple
- Each column is called as a attribute
- Domain
  - It is a set of permissible values that can be stored in an attribute
  - This feature is not available in the other models
- Relational model provides a set of operators to the user. Using these operators, the user can perform any operations on the tables.

Advantages of Relational Model

- **Insert**
  - Adding a new supplier or a part is not a problem. They are independent entities
  - If you want to represent a relation between supplier and parts, then insert a tuple in SP table.
- **Delete**
  - Delete operations are independent of other tables
- **Update**
  - Updating supplier or part information is very simple

Relational Database Management Systems (RDBMS)

- The basic functionality of RDBMS was conceptualized by Dr. E F Codd, when he was working for IBM
  - He laid down certain principles which govern the functioning of any RDBMS
- In 1974, the first standard of SQL was also developed
- Also C J Date from IBM also contributed towards standardization of RDBMS

RDBMS Terminology

- **Relation**
  - It is equivalent of a table
- **Tuple**
  - It is equivalent of a single row in a relation
- **Attribute**
  - It is equivalent of a column in a relation
- **Primary Key**
  - It is a unique identifier which identifies each tuple uniquely

The Relational Data Structure

- Each relation is made up of two parts
  - Intension
    - It is the fixed part of the relation which contains the column names
  - Extension
    - It is the data part of a relation

The Relational Data Structure

- The smallest unit of data in the relational model is the individual value
- Each value is atomic. They don’t have any internal structure as far as the relational model is concerned
- A domain is a set of all possible values which a attribute can take
- Domains are conceptual in nature.
  - They can be stored in the database as a set of values
  - Once stored in the database, they can be used in any table definition.

Degree and Cardinality of a relation

- The number of attributes in a relation is called the degree of the relation
- The number of tuples in a a relation is called the cardinality of the relation
- The cardinality of a relation changes with more addition of tuples, but the degree does not

The Relational Data Integrity

- Every relation has a Candidate Key. A candidate key is a key which can uniquely identify a tuple in a relation with n cardinality
- A candidate key should posses the following characteristic
  - Uniqueness
  - Minimality
- Every relation has at least one candidate key. This key is designated as a Primary Key
- In case if there are more than one candidate keys, then the most appropriate one is designated as the primary key and the rest are called Alternate Keys.

The Relational Data Integrity

- **Similarly, there is a concept of Foreign Key. A foreign key is a set of attributes from a table, whose values depend on the primary key of other table.**
- **In our example, in the SP relation, the values of S# depend on the values from the S relation.**

Two integrity rules in a relational database

- **Entity Integrity Rule**
  - No attribute participating in a the primary key of a base relation is allowed to contain any NULL values.
- **Referential Integrity Rule**
  - The value of the foreign key must be one of the values of the primary key (or unique alternate key) from the other table on which it is dependent OR it may contain NULL values.

Creating and Managing Tables

Naming Conventions

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

The CREATE TABLE Statement

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

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

  - Column name, column datatype, and column size

The ALTER TABLE Statement

- **Use the ALTER TABLE statement to:**
  - **Add a new column**
  - **Modify an existing column**
  - **Define a default value for the new column**

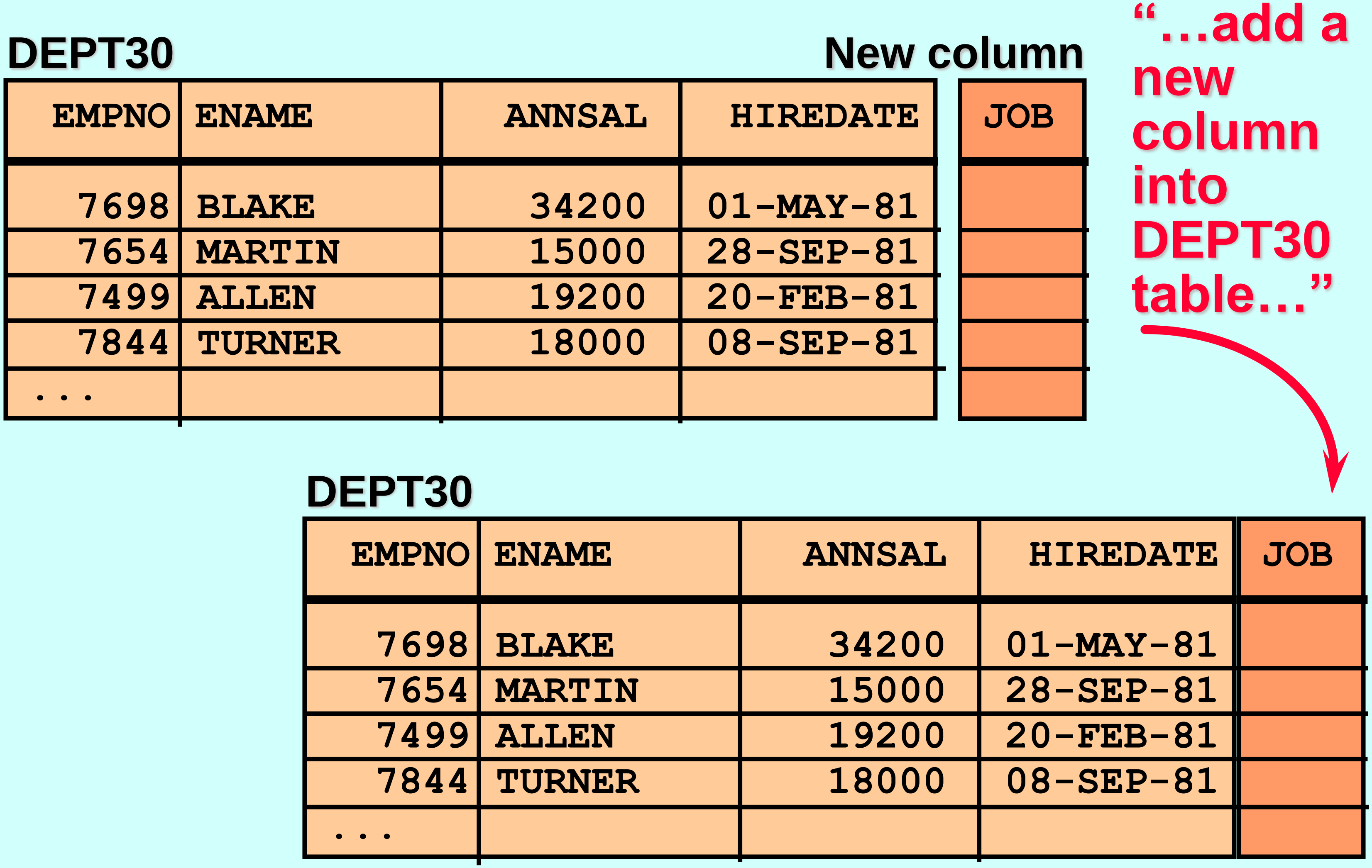
```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

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

Datatypes

Datatype	Description
VARCHAR(*size*)	Variable-length character data
CHAR(*size*)	Fixed-length character data
NUMBER(*p,s*)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data up to 2 gigabytes
CLOB	Single-byte character data up to 4 gigabytes
RAW and LONG RAW	Raw binary data
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes

Adding a Column



DEPT30

EMPNO	ENAME	ANNSAL	HIREDATE
7698	BLAKE	34200	01-MAY-81
7654	MARTIN	15000	28-SEP-81
7499	ALLEN	19200	20-FEB-81
7844	TURNER	18000	08-SEP-81
. . .			

New column

JOB

“…add a new column into DEPT30 table…”

DEPT30

EMPNO	ENAME	ANNSAL	HIREDATE	JOB
7698	BLAKE	34200	01-MAY-81	
7654	MARTIN	15000	28-SEP-81	
7499	ALLEN	19200	20-FEB-81	
7844	TURNER	18000	08-SEP-81	
. . .				

Adding a Column

- You use the ADD clause to add columns.

```
SQL> ALTER TABLE dept30
  2  ADD         (job VARCHAR(9));
Table altered.
```

- The new column becomes the last column.

```
    EMPNO ENAME          ANNSAL HIREDATE  JOB
--------- ---------- --------- --------- ----
     7698 BLAKE          34200 01-MAY-81
     7654 MARTIN         15000 28-SEP-81
     7499 ALLEN          19200 20-FEB-81
     7844 TURNER         18000 08-SEP-81
...
6 rows selected.
```

Modifying a Column

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

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

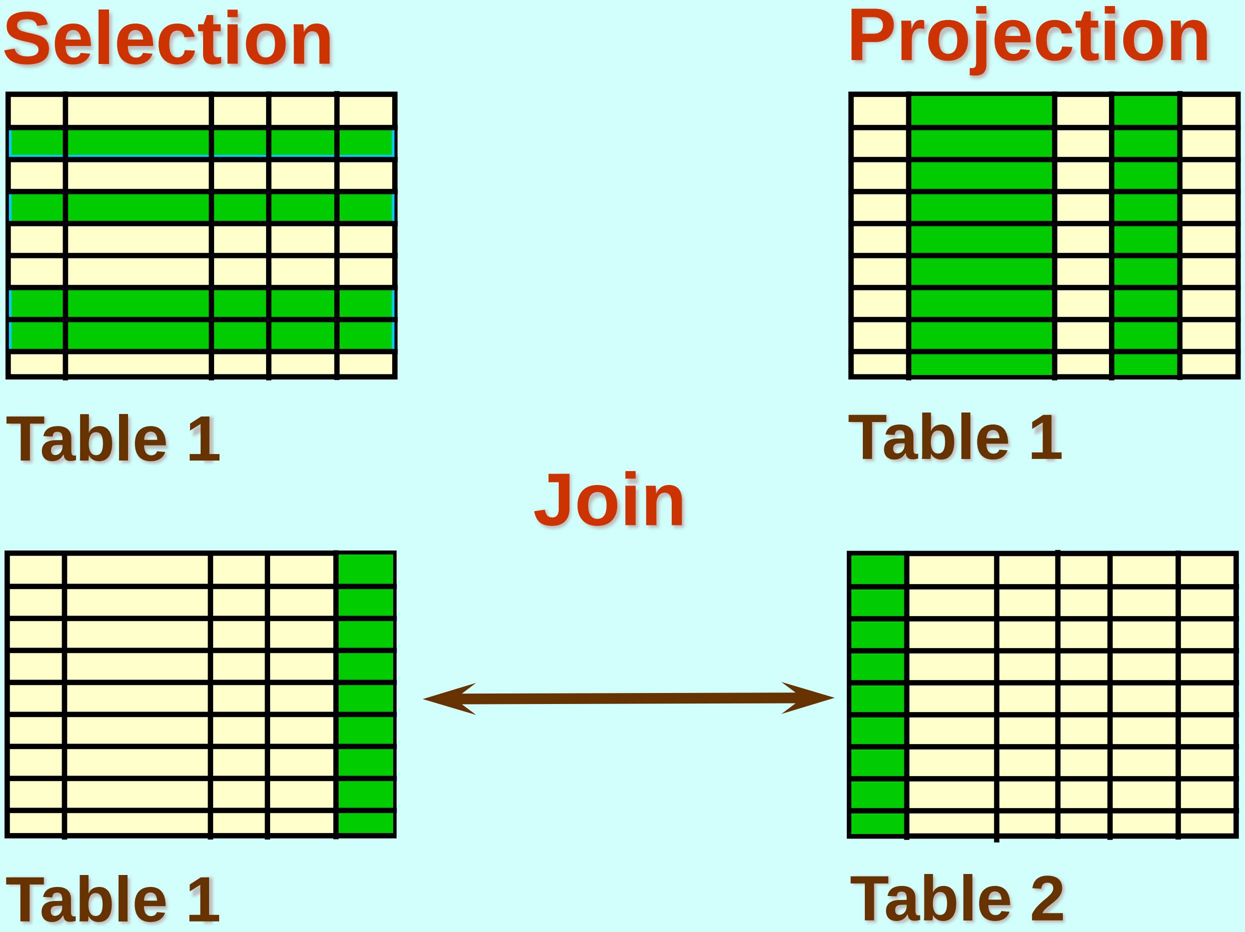
```
ALTER TABLE   dept30
MODIFY         (ename VARCHAR(15));
Table altered.
```

Dropping a Table

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

```
SQL> DROP TABLE dept30;
Table dropped.
```

Capabilities of SQL SELECT Statements



Selection

Projection

Table 1

Table 1

Join

Table 1

Table 2

Basic SELECT Statement

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

- SELECT identifies *what* columns
- FROM identifies *which* table

Writing SQL Statements

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

Selecting All Columns

```
SQL> SELECT *
  2  FROM   dept;
```

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

Selecting Specific Columns

```
SQL> SELECT   deptno, dname
  2  FROM   dept;
```

```
   DEPTNO DNAME
--------- --------------
       10 ACCOUNTING
       20 RESEARCH
       30 SALES
       40 OPERATIONS
```

Arithmetic Expressions

- Create expressions on NUMBER and DATE data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

```
SQL> SELECT ename, sal, sal+300
  2  FROM   emp;
```

```
ENAME            SAL   SAL+300
---------- --------- ---------
KING            5000      5300
BLAKE           2850      3150
CLARK           2450      2750
JONES           2975      3275
MARTIN          1250      1550
ALLEN           1600      1900
...
14 rows selected.
```

Operator Precedence

*   /   +   –

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

Operator Precedence

```
SQL> SELECT ename, sal, 12*sal+100
  2  FROM   emp;
```

```
ENAME             SAL 12*SAL+100
---------- ---------- ----------
KING             5000      60100
BLAKE            2850      34300
CLARK            2450      29500
JONES            2975      35800
MARTIN           1250      15100
ALLEN            1600      19300
...
14 rows selected.
```

Using Parentheses

```
SQL> SELECT ename, sal, 12*(sal+100)
  2  FROM   emp;
```

```
ENAME             SAL 12*(SAL+100)
---------- ---------- -----------
KING             5000       61200
BLAKE            2850       35400
CLARK            2450       30600
JONES            2975       36900
MARTIN           1250       16200
...
14 rows selected.
```

Defining a Null Value

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

```
SQL> SELECT  ename, job, comm
  2  FROM    emp;
```

```
ENAME      JOB             COMM
---------- --------- ---------
KING       PRESIDENT
BLAKE      MANAGER
...
TURNER     SALESMAN           0
...
14 rows selected.
```

Null Values in Arithmetic Expressions

- **Arithmetic expressions containing a null value evaluate to null.**

```
SQL> select ename NAME, 12*sal+comm
  2  from   emp
  3  WHERE  ename='KING';
```

```
NAME        12*SAL+COMM
----------- -----------
KING
```

Defining a Column Alias

- Renames a column heading
- Is useful with calculations
- Immediately follows column name; optional AS keyword between column name and alias
- Requires double quotation marks if it contains spaces or special characters or is case sensitive

Using Column Aliases

```
SQL> SELECT ename AS name, sal salary
  2  FROM   emp;
```

```
NAME                 SALARY
------------- ---------
...
```

```
SQL> SELECT ename "Name",
  2         sal*12 "Annual Salary"
  3  FROM   emp;
```

```
Name          Annual Salary
------------- -------------
...
```

Limiting Rows Using a Selection

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
...				

"…retrieve all employees
in department 10"

EMP

EMPNO	ENAME	JOB	...	DEPTNO
7839	KING	PRESIDENT		10
7782	CLARK	MANAGER		10
7934	MILLER	CLERK		10

Limiting Rows Selected

- Restrict the rows returned by using the WHERE clause.

- The WHERE clause follows the FROM clause.

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

Using the WHERE Clause

```
SQL> SELECT ename, job, deptno
  2  FROM   emp
  3  WHERE  job='CLERK';
```

```
ENAME      JOB          DEPTNO
---------- --------- ---------
JAMES      CLERK            30
SMITH      CLERK            20
ADAMS      CLERK            20
MILLER     CLERK            10
```

Character Strings and Dates

– Character strings and date values are enclosed in single quotation marks

```
SQL> SELECT   ename, job, deptno
  2  FROM     emp
  3  WHERE    ename = 'JAMES';
```

Comparison Operators

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

Using the Comparison Operators

```
SQL> SELECT ename, sal, comm
  2  FROM   emp
  3  WHERE  sal<=comm;
```

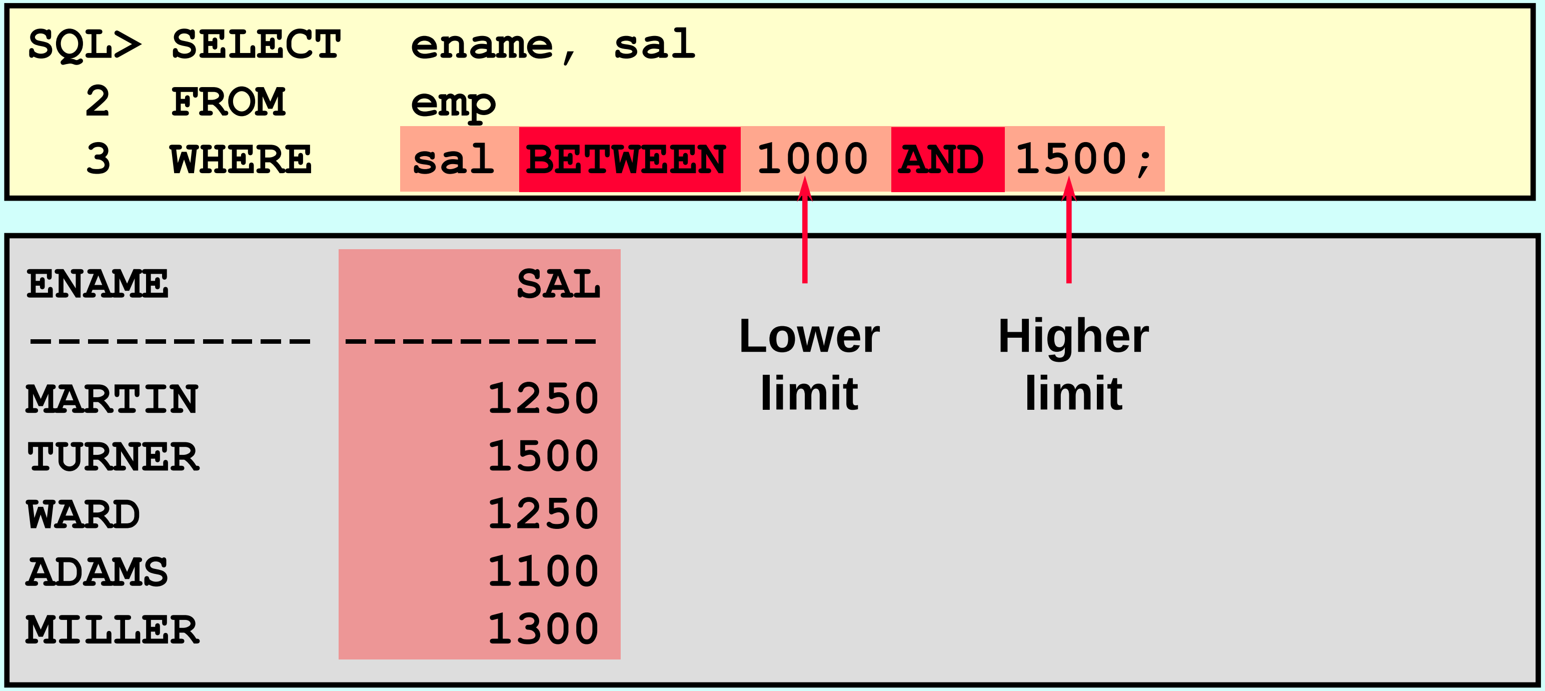
```
ENAME            SAL      COMM
---------- ---------- ----------
MARTIN          1250  ↔  1400
```

Other Comparison Operators

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

Using the BETWEEN Operator

- Use the BETWEEN operator to display rows based on a range of values.



```
SQL> SELECT   ename, sal
  2  FROM     emp
  3  WHERE    sal BETWEEN 1000 AND 1500;
```

```
ENAME             SAL
---------- ----------
MARTIN           1250
TURNER           1500
WARD             1250
ADAMS            1100
MILLER           1300
```

Using the IN Operator

- Use the IN operator to test for values in a list.

```
SQL> SELECT   empno, ename, sal, mgr
  2  FROM     emp
  3  WHERE    mgr IN (7902, 7566, 7788);
```

```
    EMPNO ENAME            SAL       MGR
--------- ---------- --------- ---------
     7902 FORD            3000      7566
     7369 SMITH            800      7902
     7788 SCOTT           3000      7566
     7876 ADAMS           1100      7788
```

Using the LIKE Operator

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

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE 'S%';
```

Using the LIKE Operator

– You can combine pattern matching characters.

```
SQL> SELECT   ename
  2  FROM     emp
  3  WHERE    ename LIKE '_A%';
```

```
ENAME
----------
JAMES
WARD
```

Using the IS NULL Operator

- Test for null values with the IS NULL operator

```
SQL> SELECT   ename, mgr
  2  FROM     emp
  3  WHERE    mgr IS NULL;
```

```
ENAME             MGR
---------- ----------
KING
```

Logical Operators

Operator	Meaning
AND	Returns TRUE if *both* component conditions are TRUE
OR	Returns TRUE if *either* component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

Using the AND Operator

AND requires both conditions to be TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  AND    job='CLERK';
```

```
    EMPNO ENAME      JOB             SAL
--------- ---------- --------- ---------
     7876 ADAMS      CLERK          1100
     7934 MILLER     CLERK          1300
```

Using the OR Operator

OR requires either condition to be TRUE.

```
SQL> SELECT empno, ename, job, sal
  2  FROM   emp
  3  WHERE  sal>=1100
  4  OR     job='CLERK';
```

```
    EMPNO ENAME      JOB             SAL
--------- ---------- --------- ---------

     7839 KING       PRESIDENT      5000
     7698 BLAKE      MANAGER        2850
     7782 CLARK      MANAGER        2450
     7566 JONES      MANAGER        2975
     7654 MARTIN     SALESMAN       1250
...
14 rows selected.
```

Using the NOT Operator

```
SQL> SELECT ename, job
  2  FROM   emp
  3  WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

```
ENAME      JOB
---------- ---------
KING       PRESIDENT
MARTIN     SALESMAN
ALLEN      SALESMAN
TURNER     SALESMAN
WARD       SALESMAN
```

Rules of Precedence

Order Evaluated	Operator
1	All comparison operators
2	NOT
3	AND
4	OR

- Override rules of precedence by using parentheses.

Rules of Precedence

```
SQL> SELECT ename, job, sal
  2  FROM   emp
  3  WHERE  job='SALESMAN'
  4  OR     job='PRESIDENT'
  5  AND    sal>1500;
```

```
ENAME      JOB             SAL
---------- --------- ---------
KING       PRESIDENT      5000
MARTIN     SALESMAN       1250
ALLEN      SALESMAN       1600
TURNER     SALESMAN       1500
WARD       SALESMAN       1250
```

Rules of Precedence

Use parentheses to force priority.

```
SQL> SELECT    ename, job, sal
  2  FROM      emp
  3  WHERE     (job='SALESMAN'
  4  OR        job='PRESIDENT')
  5  AND       sal>1500;
```

```
ENAME      JOB             SAL
---------- --------- ---------
KING       PRESIDENT      5000
ALLEN      SALESMAN       1600
```

ORDER BY Clause

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

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate;
```

```
ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
SMITH      CLERK            20 17-DEC-80
ALLEN      SALESMAN         30 20-FEB-81
...
14 rows selected.
```

Sorting in Descending Order

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate DESC;
```

```
ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
ADAMS      CLERK            20 12-JAN-83
SCOTT      ANALYST          20 09-DEC-82
MILLER     CLERK            10 23-JAN-82
JAMES      CLERK            30 03-DEC-81
FORD       ANALYST          20 03-DEC-81
KING       PRESIDENT        10 17-NOV-81
MARTIN     SALESMAN         30 28-SEP-81
...
14 rows selected.
```

Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table

- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Adding a New Row to a Table

50	DEVELOPMENT	DETROIT

New row

DEPT

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

"…insert a new row into DEPT table…"

DEPT

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

The INSERT Statement

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

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

Inserting New Rows

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

- Enclose character and date values within single quotation marks.

```
SQL> INSERT INTO     dept (deptno, dname, loc)
  2  VALUES           (50, 'DEVELOPMENT', 'DETROIT');
1 row created.
```

Changing Data in a Table

EMP

EMPNO	ENAME	JOB	. . .	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		10
7566	JONES	MANAGER		20
. . .				

"…update a row in EMP table…"

EMP

EMPNO	ENAME	JOB	. . .	DEPTNO
7839	KING	PRESIDENT		10
7698	BLAKE	MANAGER		30
7782	CLARK	MANAGER		20
7566	JONES	MANAGER		20
. . .				

The UPDATE Statement

– Modify existing rows with the UPDATE statement.

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

Updating Rows in a Table

- – Specific row or rows are modified when you specify the WHERE clause.
- – All rows in the table are modified if you omit the WHERE clause.

```
SQL> UPDATE   emp
  2  SET      deptno = 20
  3  WHERE    empno = 7782;
1 row updated.
```

```
SQL> UPDATE   employee
  2  SET      deptno = 20;
14 rows updated.
```

Removing a Row from a Table

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	DEVELOPMENT	DETROIT
60	MIS	
. . .		

"…delete a row from DEPT table…"

DEPT

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

The DELETE Statement

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

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

Deleting Rows from a Table

- Specific row or rows are deleted when you specify the WHERE clause.
  All rows in the table are deleted if you omit the WHERE clause.

```
SQL> DELETE FROM    department
  2  WHERE          dname = 'DEVELOPMENT';
1 row deleted.
```

```
SQL> DELETE FROM    department;
4 rows deleted.
```