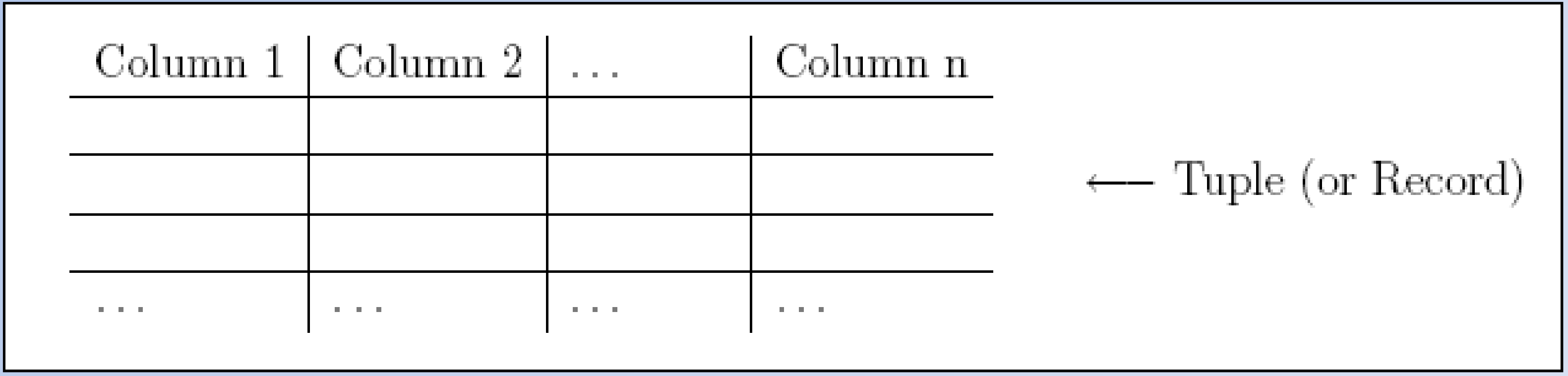
SQL – Structured Query Language

Introduction

Tables

- In relational database systems data are represented using tables (relations).
- A query issued against the database also results in a table.
- A table has the following structure:



Column 1	Column 2	...	Column n
...	...	...	...

$\longleftarrow$ Tuple (or Record)

- SQL uses the terms table, row, and column for relation, tuple, and attribute, respectively.

Basic data types

- **char(n)**:
  - Fixed-length character data (string), n characters long.
  - The maximum size for n is 2000 bytes.
  - Note that a string of type char is always padded on right with blanks to full length of n. (memory consuming)
  - Example: char(40)
- **varchar2(n):**
  - Variable-length character string.
  - The maximum size for n is 4000 bytes.
  - Only the bytes used for a string require storage.
  - Example: varchar2(80)

Basic data types

- **number(o, d):**
  - Numeric data type for integers and reals.
    - o is the ***precision***= total number of significant digits
    - d is the ***scale*** = the number of digits from the decimal point to the least significant digit
  - Maximum values: o =38, d= −84 to +127.
  - Examples: number(8), number(5,2)
  - Note that, number(5,2) cannot contain anything larger than 999.99 without resulting in an error.
  - Data types derived from number are **int[eger]**, **dec[imal]**, **smallint**, **float** and **real**.

Number format

- **number(o, d):**
  - positive scale (d) is the number of digits to the right of the decimal point including the least significant digit which is the right-most digit after the decimal point
  - negative scale (d) is the number of significant digits to the left of the decimal point, to but not including the least significant digit.
  - For positive scale (d) the least significant digit actual data is rounded to the specified number of places to the right or left of the decimal point.
  - For example, a specification of (10,-2) means to round to hundreds.

Number examples

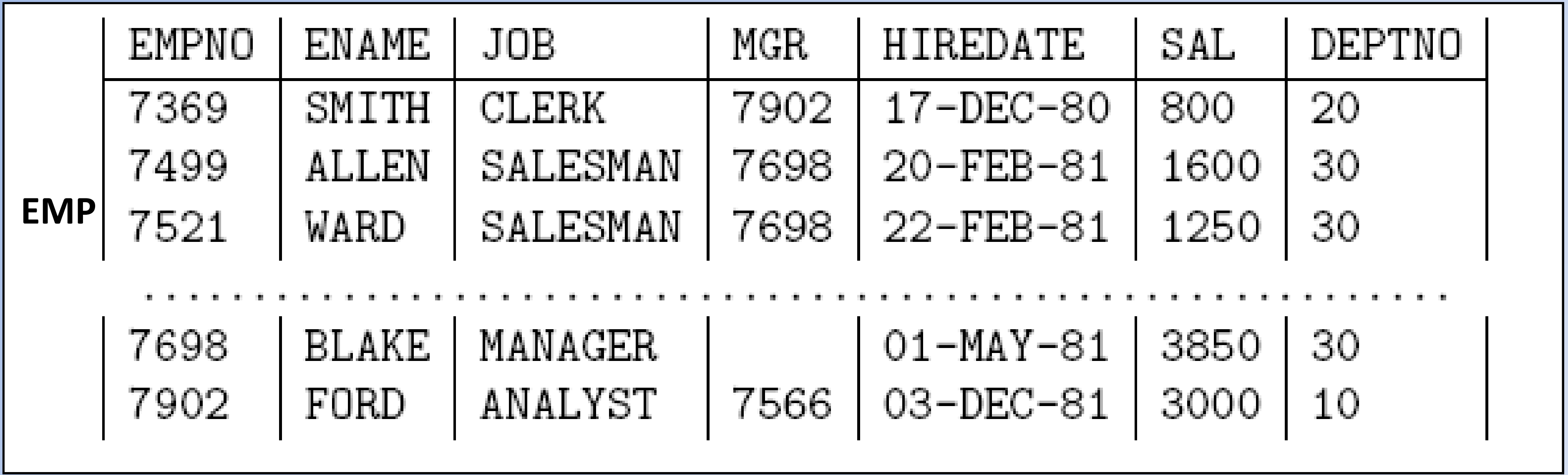
- The value *7,456,123.89* will display as follows:

– number(9)	*7456124*
– number(9,1)	*7456123.9*
– number(*,1)	*7456123.9*
– number(9,2)	*7456123.89*
– number(6)	*[not accepted exceeds precision]*
– number(7,-2)	*7456100*
– number	*7456123.89*
– float	*7456123.89*
– float(12)	*7456000.0*

Basic data types

- **date**:
  - Date data type for storing date and time.
  - The default format for a date is: DD-MMM-YY.
  - Examples: ’13-OCT-94’, ’07-JAN-98’
- **long**:
  - Character data up to a length of 2GB.
- In Oracle-SQL there is no data type boolean. It can, however, be simulated by using either char(1) or number(1).

Example Tables



EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	30
....						
7698	BLAKE	MANAGER		01-MAY-81	3850	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	10

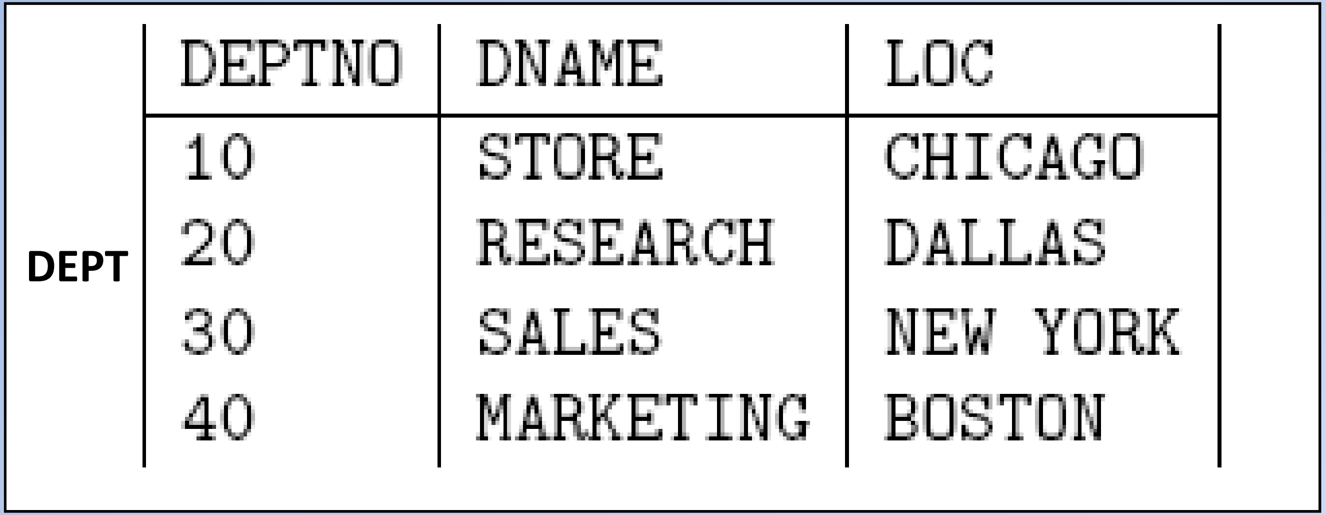
EMPNO: number(4) ***ENAME***: varchar2(30)

JOB: char(10) ***MGR***: number(4)

HIREDATE: date ***SAL***: number(7,2)

DEPTNO: number(2)

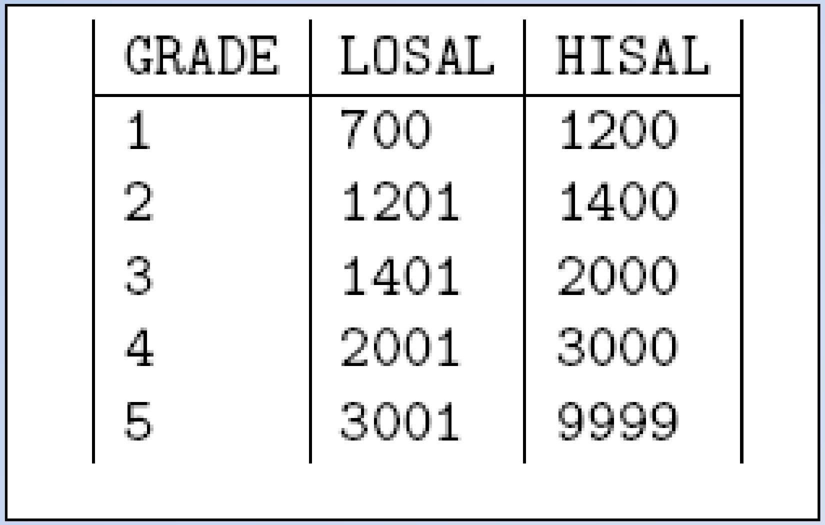
Example Tables



DEPT

DEPTNO	DNAME	LOC
10	STORE	CHICAGO
20	RESEARCH	DALLAS
30	SALES	NEW YORK
40	MARKETING	BOSTON

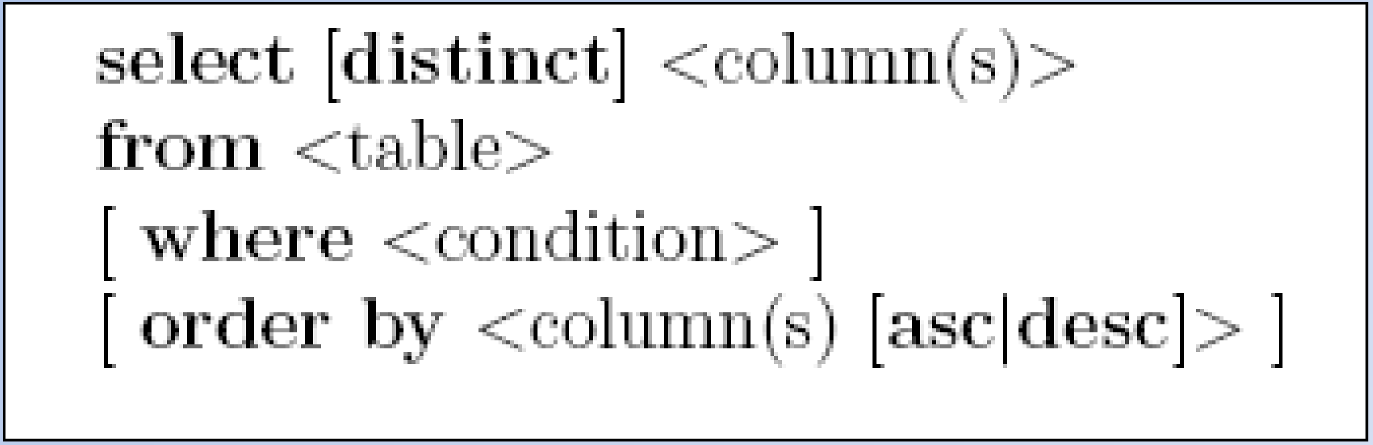
SALGRADE



GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

Queries

- In SQL a query has the following (simplified) form (components in brackets [ ] are optional):



```
select [distinct] <column(s)>
from <table>
[ where <condition> ]
[ order by <column(s) [asc|desc]> ]
```

Queries

- The columns to be selected from a table are specified after the keyword select. This operation is also called projection.
  - ***select LOC, DEPTNO from DEPT***
- If all columns should be selected, the asterisk symbol ‘*’ can be used to denote all attributes.
  - ***select * from EMP***
- Instead of an attribute name, the select clause may also contain arithmetic expressions involving arithmetic operators etc.
  - ***select ENAME, DEPTNO, SAL*1.55 from EMP***

Queries : distinct

- Consider the query
  - ***select DEPTNO from EMP***
- It retrieves the department number for each tuple.
- Typically, some numbers will appear more than only once in the query result, that is, duplicate result tuples are not automatically eliminated.
- Inserting the keyword distinct after the keyword select, however, forces the elimination of duplicates from the query result.
  - ***select distinct DEPTNO from EMP***

Queries : order by

- It is also possible to specify a sorting order in which the result tuples of a query are displayed.
- For this the order by clause is used and which has one or more attributes listed in the select clause as parameter. *desc* specifies a descending order and *asc* specifies an ascending order (this is also the default order).
- For example,
  - ***select ENAME, DEPTNO, HIREDATE from EMP***

    order by DEPTNO [asc], HIREDATE desc

Queries : where clause

- If one is interested in tuples that satisfy certain conditions, the where clause is used.
- List the job title and the salary of those employees who earn more than 1500:
  - ***select JOB, SAL from EMP where SAL > 1500***
- List the job title and the salary of those employees who earn more than 1500 and whose manager has the number 7698 or 7566
  - ***select JOB, SAL from EMP where (MGR = 7698 or MGR = 7566) and SAL > 1500***

Queries : where clause

- For all data types, the comparison operators =, != or <>,<, >,<=, => are allowed in the conditions of a where clause.
- Further comparison operators are:
- **Set Conditions**: <column> [not] in (<list of values>)
  - ***select * from DEPT where DEPTNO in (20,30)***
- **Null value**: <column> is [not] null,
- i.e., for a tuple to be selected there must (not) exist a defined value for this column.
  - ***select * from EMP where MGR is not null***
- Note: the operations = null and ! = null are not defined!

Queries : where clause

- **Domain conditions**: <column> [not] between <lower bound> and <upper bound>
  - ***select EMPNO, ENAME, SAL from EMP where SAL between 1500 and 2500***
  - ***select ENAME from EMP where HIREDATE between '02-APR-81' and '08-SEP-81'***

String

- We know that in order to compare an attribute with a string, it is required to surround the string by quotes
  - ***select * from DEPT where DNAME='SALES'***
- A powerful operator for pattern matching is the ***like*** operator.
- Together with this operator, two special characters are used
  - percent sign % (wild card) & underline _ (position marker)
- Find the employees whose name starts with 'S'.
  - ***select * from EMP where ENAME like 'S%'***
- Find the employees whose name starts with 'S' and ends with 'T'
  - ***select * from EMP where ENAME like 'S%T'***

String

- Find all tuples of the table DEPT that contain two C in the name of the department
  - ***select * from DEPT where DNAME like ’%C%C%’***
- Find all tuples of the table DEPT that contain exactly one character appears between the two Cs .
  - ***select * from DEPT where DNAME like ’%C_C%’***

String

- *upper*(string) : upper(‘aBcd’) – ‘ABCD’
- *lower*(string) : lower(‘aBcd’) – ‘abcd’
- *initcap*(string) : initcap(‘aBcd’) – ‘Abcd’
- *length*(string) : length(‘abcd’) - 4
- *substr*(string, start, [n]) : substr(‘abcdefgh’,2,4) - bcde
- *lpad*(string,length,[‘chars’]) : lpad(‘ha’,5,’a’) - aaaha
- *rpad*(string,length,[‘chars’]) : rpad(‘ha’,5,’a’) - haaaa
- *ltrim*(string, [‘chars’]) : ltrim(‘abracadabra’,’ab’)- ‘racadabra’
- *rtrim*(string, [‘chars’]) : rtrim(‘abracadabra’,’ab’)- ‘abracadabr’

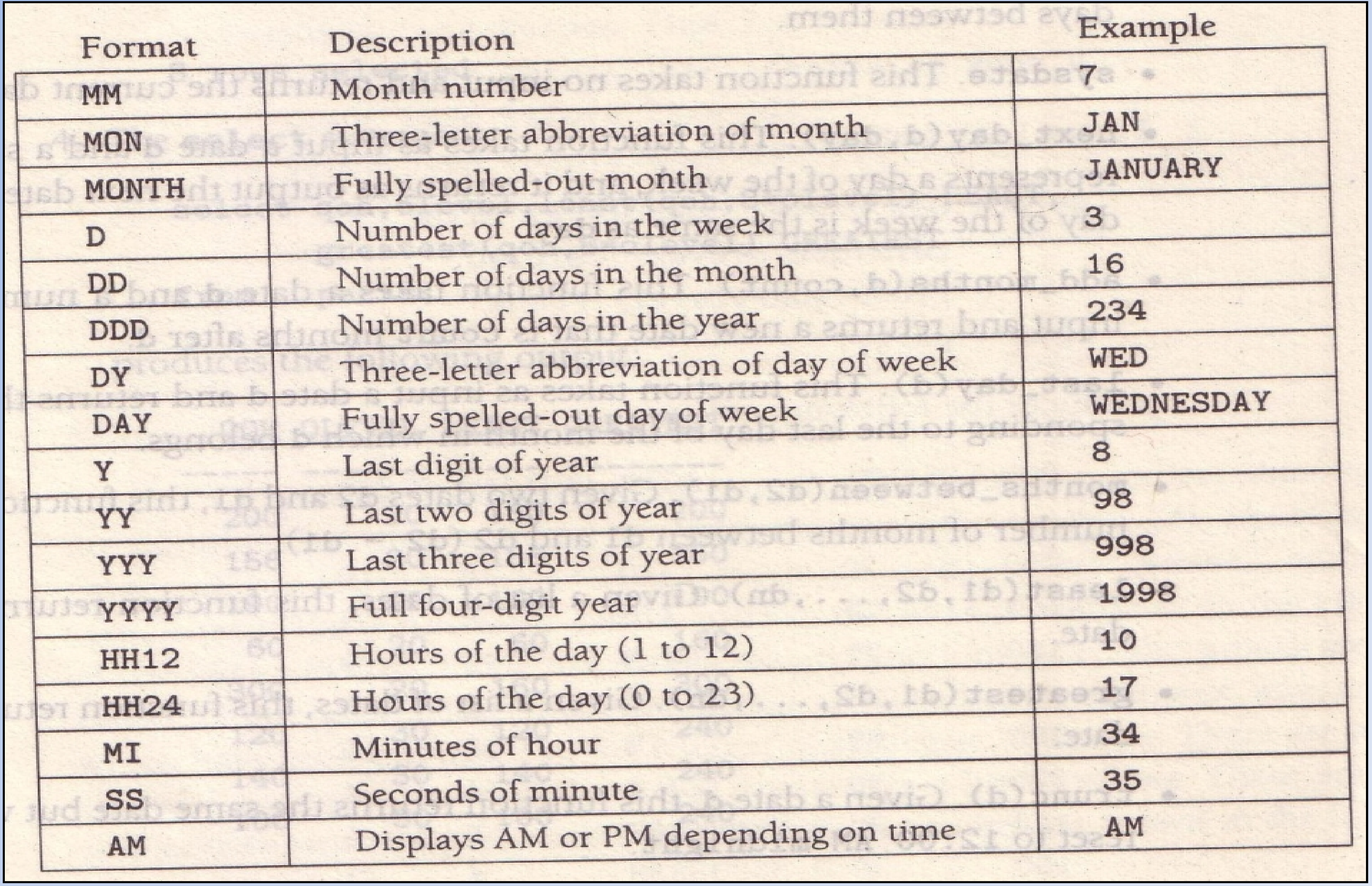
String

- *instr*(string, ‘chars’[,start [,n]]) : instr(‘abracadabra’,’cad’) – 5
  - ***select * from EMP where instr(ENAME,’John’) > 0***
- String concatenation
  - can be done using ||
  - ***select EMPNO|| ‘, ’ || ENAME from EMP***
- One more thing, any query result’s column can be renamed to any other name. This is known as alias.
  - ***select EMPNO|| ‘, ’ || ENAME as EID from EMP***

Date

- Oracle’s default format is ‘DD-MON-YY’
- *Sysdate* - returns the current date
  - ***select sysdate from dual***
- *to_date* - returns a date
  - ***select to_date(’12-01-2001’,’DD-MM-YYYY’) from dual***
- *to_char* - returns a string
  - ***select to_char(sysdate,’DD-MON-YY, HH:MI:SS’) from dual***
- *to_numbe**r –*** returns a number
  - ***select to_number(‘1234’) from dual***

Date Formats



Format	Description	Example
MM	Month number	7
MON	Three-letter abbreviation of month	JAN
MONTH	Fully spelled-out month	JANUARY
D	Number of days in the week	3
DD	Number of days in the month	16
DDD	Number of days in the year	234
DY	Three-letter abbreviation of day of week	WED
DAY	Fully spelled-out day of week	WEDNESDAY
Y	Last digit of year	8
YY	Last two digits of year	98
YYY	Last three digits of year	998
YYYY	Full four-digit year	1998
HH12	Hours of the day (1 to 12)	10
HH24	Hours of the day (0 to 23)	17
MI	Minutes of hour	34
SS	Seconds of minute	35
AM	Displays AM or PM depending on time	AM

Aggregation

- ***Count*** - Counting Rows
- How many tuples are stored in the relation EMP?
  - ***select count(*) from EMP***
- How many different job titles are stored in the relation EMP?
  - ***select count(distinct JOB) from EMP***
- ***Sum*** - Computes the sum of values (only applicable to the data type number)
- Find the sum of all salaries of employees working in the department 30.
  - ***select sum(SAL) from EMP where DEPTNO = 30***

Aggregation

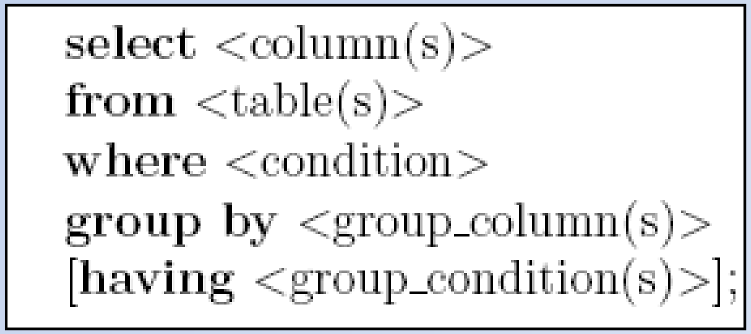
- ***Max/Min*** – Maximum/Minimum value for a column
- List the minimum and maximum salary.
  - ***select min(SAL), max(SAL) from EMP***
- Compute the difference between the minimum and maximum salary.
  - ***select max(SAL) - min(SAL) as difference from EMP***
- ***Avg*** - Computes average value for a column (only applicable to the data type number)
- Find the average salaries of employees working in the department 10.
  - ***select avg(SAL) from EMP where DEPTNO = 10***
-

Aggregation

- Ignores tuples that have a null value for the specified attribute.
- It is not possible to use aggregation of aggregation. So max(avg(..)) is not possible.
- Aggregation can be placed in sub query.
- Find the name of the employee with maximum salary.
  - ***select ENAME from EMP where SAL = (select max(SAL) from EMP)***

Group By

- We have seen how aggregate functions can be used to compute a single value for a column.
- Often applications require grouping rows that have certain properties and then applying an aggregate function on one column for each group separately.
- For this, SQL provides the clause group by <group column(s)>. This clause appears after the where clause and must refer to columns of tables listed in the from clause.



```
select <column(s)>
from <table(s)>
where <condition>
group by <group_column(s)>
[having <group_condition(s)>];
```

Group By

- Those rows retrieved by the selected clause that have the same value(s) for <group column(s)> are grouped.
- Aggregations specified in the select clause are then applied to each group separately.
- It is important that only those columns that appear in the <group column(s)> clause can be listed without an aggregate function in the select clause.
- For each department, we want to retrieve the minimum and maximum salary.
  - ***select DEPTNO, min(SAL), max(SAL) from EMP group by DEPTNO***

Group By

- Rows to form a group can be restricted in the where clause. For example, if we add the condition ***where JOB = ’CLERK’,*** only respective rows build a group. The query then would retrieve the minimum and maximum salary of all clerks for each department.
- Note that is not allowed to specify any other column than DEPTNO without an aggregate function in the select clause since this is the only column listed in the group by clause.

Group By/Having

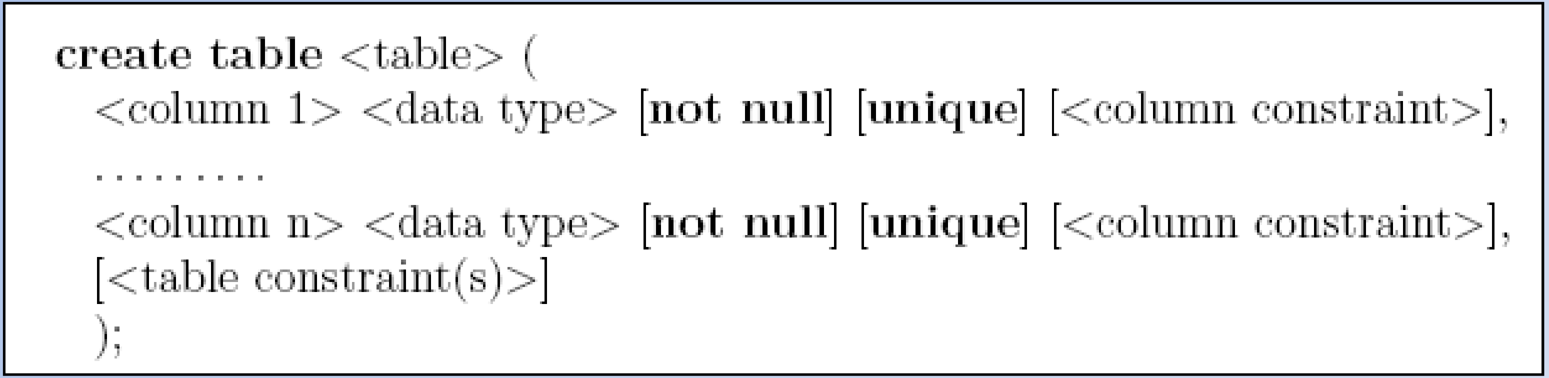
- Once groups have been formed, certain groups can be eliminated based on their properties, e.g., if a group contains less than three rows.
- This type of condition is specified using the having clause. As for the select clause also in a having clause only <group column(s)> and aggregations can be used.
- Retrieve the minimum and maximum salary of clerks for each department having more than three clerks.
  - ***select DEPTNO, min(SAL), max(SAL) from EMP where JOB = 'CLERK' group by DEPTNO having count(*) > 3***
- Note that it is even possible to specify a sub query in a having clause.

Group By/Having

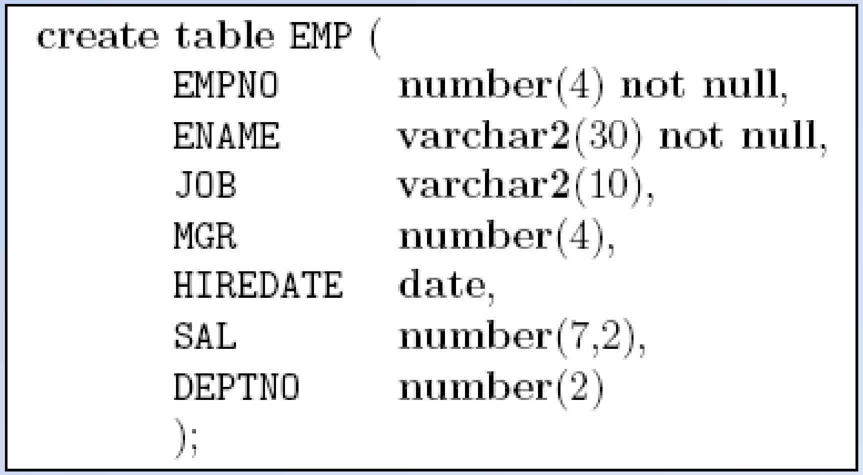
- A query containing a group by clause is processed in the following way:
  - Select all rows that satisfy the condition specified in the where clause.
  - From these rows form groups according to the group by clause.
  - Discard all groups that do not satisfy the condition in the having clause.
  - Apply aggregate functions to each group.
  - Retrieve values for the columns and aggregations listed in the select clause.

Create Table

- The SQL command for creating an empty table has the following form:



```
create table <table> (
  <column 1> <data type> [not null] [unique] [<column constraint>],
  .........
  <column n> <data type> [not null] [unique] [<column constraint>],
  [<table constraint(s)>]
  );
```



```
create table EMP (
        EMPNO       number(4) not null,
        ENAME       varchar2(30) not null,
        JOB         varchar2(10),
        MGR         number(4),
        HIREDATE    date,
        SAL         number(7,2),
        DEPTNO      number(2)
        );
```

Constraints

- The definition of a table may include the specification of integrity constraints.
- Basically two types of constraints are provided:
  - column constraints are associated with a single column
  - table constraints are typically associated with more than one column.
- The specification of a (simple) constraint has the following form:

  [constraint <name>] primary key | unique | not null

Constraints

- A primary key constraint enables a unique identification of each tuple in a table.
- Based on a primary key, the database system ensures that no duplicates appear in a table.

  create table EMP (

  EMPNO number(4) constraint pk_emp primary key, . . .)
- It defines the attribute EMPNO as the primary key for the table. Each value for the attribute EMPNO thus must appear only once in the table EMP.
- A table may only have one primary key and null values are not allowed.

Sample Table Creation

- We want to create a table called **PROJECT** to store information about projects. For each project, we want to store
  - the number and the name of the project
  - the employee number of the project's manager
  - the number of persons working on the project, and
  - the budget and the start and end date of the project.
- Furthermore, we have the following conditions:
  - a project is identified by its project number
  - the name of a project must be unique
  - the manager and the budget must be defined.

Sample Table Creation

```
create table PROJECT
(
    PNO number(3) constraint prj_pk primary key,
    PNAME varchar2(60) unique,
    PMGR number(4) not null,
    PERSONS number(5),
    BUDGET number(8,2) not null,
    PSTART date,
    PEND date
)
```

Default

- Instead of a not null constraint it is sometimes useful to specify a default value for an attribute if no value is given (for example: when a tuple is inserted)
- For this, we use the default clause
- For example: If no start date is given when inserting a tuple into the table PROJECT, the project start date should be set to January 1st, 1995:

PSTART date default('01-JAN-95')

Data Modifications : Insertion

- The most simple way to insert a tuple into a table is to use the insert statement

```
insert into <table> [(<column i, . . . , column j>)]
values (<value i, . . . , value j>);
```

insert into PROJECT(PNO, PNAME, PERSONS, BUDGET, PSTART) values(313, ’DBS’, 4, 150000.42, ’10-OCT-94’)

or

insert into PROJECT values(313, ’DBS’, 7411, null, 150000.42, ’10-OCT-94’, null)

Data Modifications : Insertion

- If there are already some data in other tables, these data can be used for insertions into a new table.
- For this, we write a query whose result is a set of tuples to be inserted.
- Such an insert statement has the form

```
insert into <table> [(<column i, . . . , column j>)] <query>
```

create table OLDEMP (

ENO number(4) not null,

HDATE date)

Data Modifications : Insertion

- We now can use the table EMP to insert tuples into this new relation:

  insert into OLDEMP (ENO, HDATE)

  select EMPNO, HIREDATE from EMP where HIREDATE < ’31-DEC-60’

Data Modifications: Update

- For modifying attribute values of (some) tuples in a table, we use the update statement:

```
update <table> set
<column i> = <expression i>, ..., <column j> = <expression j>
[where <condition>];
```

- An update statement without a where clause results in changing respective attributes of all tuples in the specified table.

Data Modifications: Update

- The employee JONES is transferred to the department 20 as a manager and his salary is increased by 1000:
  - ***update EMP set JOB = ’MANAGER’, DEPTNO = 20, SAL = SAL +1000 where ENAME = ’JONES’***
- All employees working in the departments 10 and 30 get a 15% salary increase.
  - ***update EMP set SAL = SAL*1.15 where DEPTNO in (10,30)***

Data Modifications : Delete

- All or selected tuples can be deleted from a table using the delete command:

```
delete from <table> [where <condition>];
```

- If the where clause is omitted, all tuples are deleted from the table.
- Delete all projects (tuples) that have been finished before the actual date (system date).
  - ***delete from PROJECT where PEND < sysdate***
- sysdate is a function in SQL that returns the system date.