PRACTICE ON MYSQL

ALTER TABLE AP_for
adding constraint, modifying column sizes, setting NULL/NOT NULL, DEFAULT values

- Adding CHECK constraints to enforce business rules

```
alter table AP_EMP add constraint C_EMP_EMPNO check (EMPNO between 1000 and 9999);

alter table AP_PROJECT add constraint C_PROJECT_PROJID check (PROJID between 100 and 999);

alter table AP_DEPT add constraint C_DEPT_DEPTNO check (DEPTNO between 10 and 99);

alter table AP_EMP add constraint C_EMP_ELNAME check (ELNAME=upper(ELNAME));
```

- Test constraint enforcements

```
insert into AP_EMP values (10001, 'Robert', 'TEMP',7369,sysdate()- interval 40 day,1000,null,10);

insert into AP_EMP values (9999, 'Robert', 'TEMP',7369,sysdate()-interval 40 day,1000,null,10);
```

- Modify column to add DEFAULT value

```
alter table AP_PROEMP alter HOURS set default 0 ;
```

- Modify column to allow for NULL

```
alter table AP_EMP modify job  varchar(30) NULL;
```

- Modify column to change size and set NOT NULL

```
alter table AP_EMP modify job varchar(20) NOT NULL;
```

- Adding foreign key constrain

```
alter table AP_EMP add constraint FK_EMP_MGR foreign key (MGR) references AP_EMP(EMPNO);
```

ALTER TABLE AP_to add/remove column

```
alter table AP_EMP add BIRTHDATE date;
alter table AP_EMP add ADDRESS varchar (50);

alter table AP_EMP drop column BIRTHDATE, drop column ADDRESS;
```

Creating table from other table

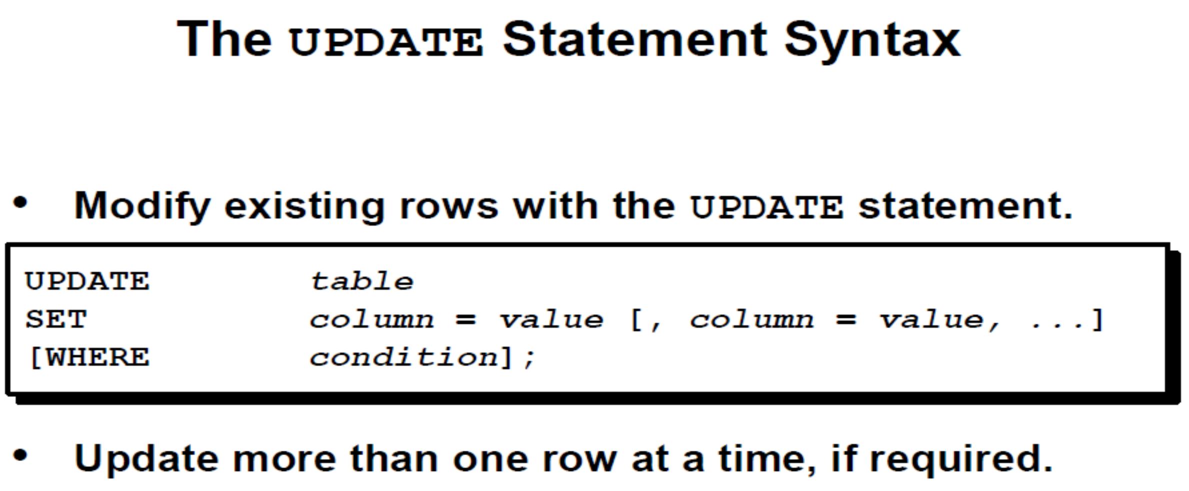
```
CREATE TABLE AP_EMPTEST AS SELECT * FROM AP_EMP -- CREATE  A TABLE FROM EXISITNG TABLE

SELECT * FROM AP_EMPTEST --  RETRIVING/SELECTING  ALL RECORDS OF A TABLE

TRUNCATE TABLE AP_EMPTEST  --  REMOVING/DELETING ALL RECORDS OF A TABLE

SELECT * FROM AP_EMPTEST-- RETRIVING/SELECTING  ALL RECORDS OF A TABLE

DROP TABLE AP_EMPTEST --  DROPPING A TABLE

SELECT * FROM AP_EMPTEST -- RETRIVING/SELECTING  ALL RECORDS OF A TABLE
```

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.

```
– CREATING A EMPTY TABLE FROM ANOTHER TABLE;
CREATE TABLE AP_DEPTTEST AS SELECT * FROM AP_DEPT WHERE 1=2;
CREATE TABLE AP_EMPTEST AS SELECT * FROM AP_EMP WHERE 1=2;

--POPULATING TABLE FROM ANOTHER TABLE;
INSERT INTO AP_DEPTTEST SELECT * FROM AP_DEPT;
INSERT INTO AP_EMPTEST SELECT * FROM AP_EMP;
COMMIT;

INSERT INTO AP_DEPTTEST (deptno,dname,loc) VALUES ('TRAINING','AUSTIN');
```

```sql
INSERT INTO AP_DEPTTEST (deptno,dname,loc) VALUES (NULL,'TRAINING','AUSTIN');
INSERT INTO AP_DEPTTEST (deptno,dname,loc) VALUES (400,'TRAINING','AUSTIN');
INSERT INTO AP_DEPTTEST  VALUES (50,'TRAINING','AUSTIN');
SELECT * FROM AP_DEPTTEST;

COMMIT;

INSERT INTO AP_DEPTTEST VALUES (60,'LEGAL',NULL);
SELECT * FROM AP_DEPTTEST;
```



The UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement.

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

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

```sql
SELECT deptno
FROM ap_emp
WHERE empno=7369;

UPDATE ap_emptest
SET deptno=30
WHERE empno=7369;
```

(IF THIS GENERATED AN ERROR IN SQL WORKBOOK: Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

TRY: SET SQL_SAFE_UPDATES = 0;)

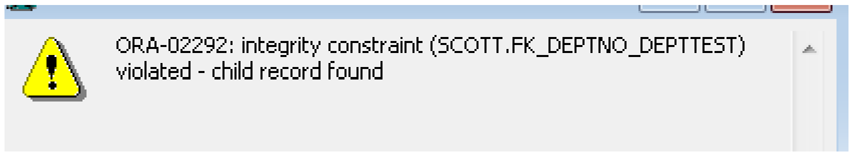
```
SELECT empno,deptno
FROM ap_emptest;

UPDATE ap_emptest
SET sal=sal+100
WHERE job='CLERK';

UPDATE ap_emptest
SET comm=comm +100;

UPDATE ap_emptest
SET comm=IFNULL(comm,0)+100;

SELECT * FROM ap_emptest;

commit;
```

The DELETE Statement

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

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

```
DELETE FROM depttest;
```


ORA-02292: integrity constraint (SCOTT.FK_DEPTNO_DEPTTEST) violated - child record found

```
DELETE FROM ap_emptest
WHERE deptno=30;

DELETE FROM ap_emptest;

SELECT COUNT(*) FROM ap_emptest;
```

SQL Statements

Oracle SQL complies with industry-accepted standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Standards Organization (ISO). Both ANSI and ISO have accepted SQL as the standard language for relational databases.

Statement	**Description**
`SELECT`	Retrieves data from the database

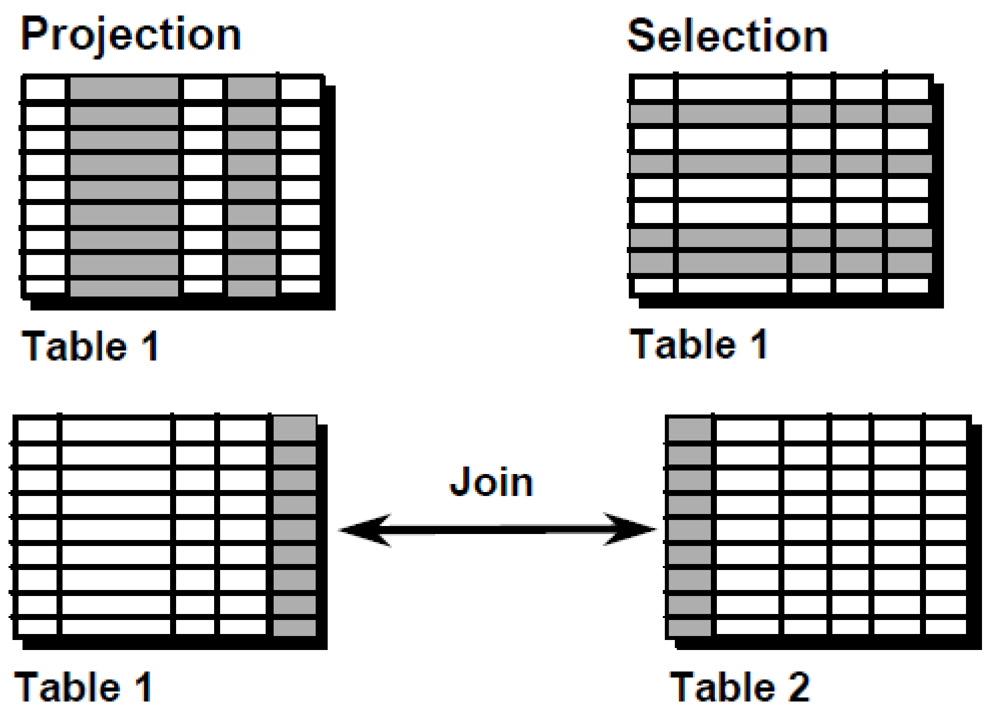
`INSERT`
`UPDATE`
`DELETE`
`MERGE`
	Enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as *data manipulation language* (DML).

`CREATE`
`ALTER`
`DROP`
`RENAME`
`TRUNCATE`
	Set up, change, and remove data structures from tables. Collectively known as *data definition language* (DDL).

`COMMIT`
`ROLLBACK`
`SAVEPOINT`
	Manage the changes made by DML statements. Changes to the data can be grouped together into logical transactions.

`GRANT` `REVOKE`	Give or remove access rights to both the Oracle database and the structures within it. Collectively known as *data control language* (DCL).

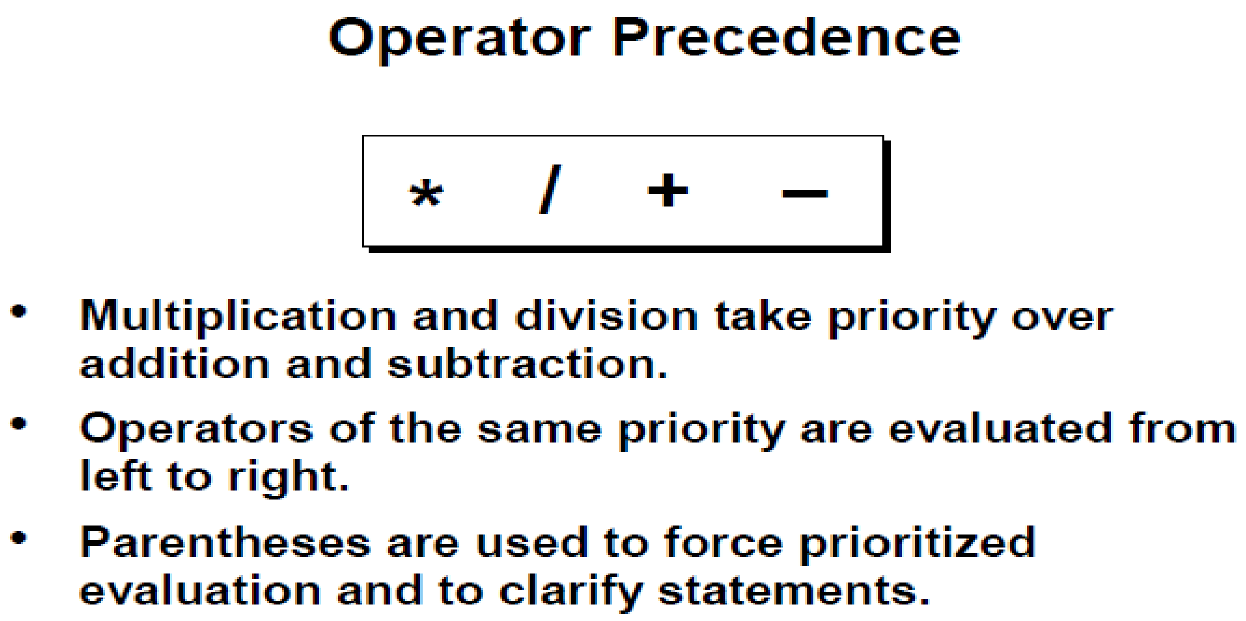
Capabilities of SQL `SELECT` Statements



<u>Basic SELECT structure</u>

SELECT	**< column list, expressions, literals>**
FROM	**<table list>**
WHERE	**<filter conditions with AND/OR/NOT logical operators)**
GROUP BY	**<column list for aggregate functions COUNT/SUM/MIN/MAX/AVG etc.>**
HAVING	**<filter conditions for grouping results>**
ORDER BY	**<column list for sorting result set>**

(SELECT and FORM clause are mandatory, all other clauses are optional and to be used as required by query result)



Operator Precedence

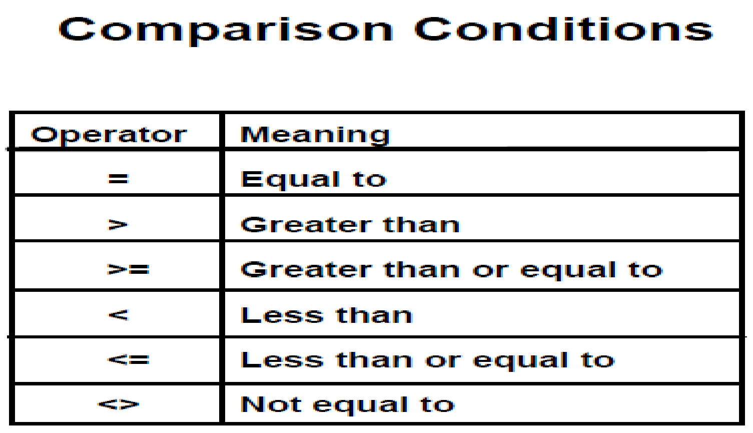
*	/	+	–

- 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.

```
SELECT *
FROM ap_emp;

SELECT empno, ELNAME, job, sal
FROM ap_emp;

SELECT empno "Employee Number", ELNAME as name , job, sal  "Monthly Salary USD"
FROM ap_emp;

SELECT A.ELNAME, A.sal, A.sal+100, A.comm, A.comm+10
from ap_emp A;

SELECT A.ELNAME, A.sal, A.sal+100, A.comm, IFNULL(A.comm,0)+10
FROM ap_emp A;

SELECT DISTINCT job
FROM ap_emp;

SELECT DISTINCT job,deptno
FROM ap_emp;

SELECT DISTINCT job,deptno
FROM ap_emp
ORDER BY job, deptno;
```

```
SELECT DISTINCT job,deptno
FROM ap_emp
ORDER BY deptno, job;

SELECT DISTINCT job,deptno
FROM ap_emp
ORDER BY deptno, job desc;

SELECT ELNAME, sal
FROM ap_emp
ORDER BY 2;

SELECT ELNAME, sal
FROM ap_emp
ORDER BY 2 desc, 1;
```

WHERE clause (SELECTION of results - Limiting result sets based upon the conditions)

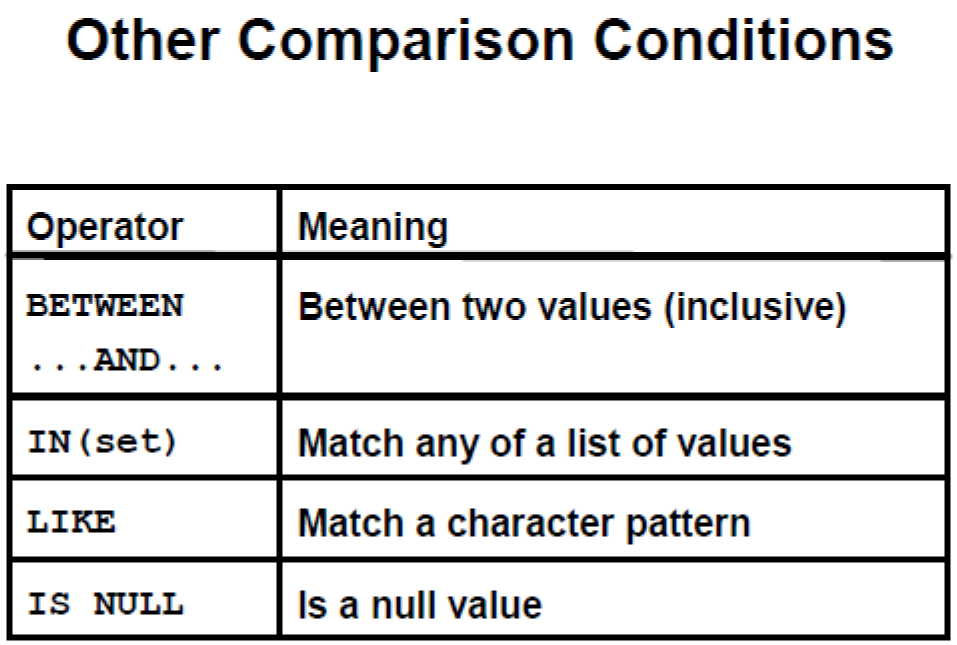


Comparison Conditions

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

```
SELECT *
FROM ap_emp
WHERE sal>9000;

SELECT *
FROM ap_emp
WHERE sal<9000;

SELECT ELNAME, sal AS salary
FROM ap_emp
WHERE sal>=9000
ORDER BY salary;

SELECT ELNAME, deptno
FROM ap_emp
WHERE deptno=10
ORDER BY 1;
```

```
SELECT ELNAME, deptno
FROM ap_emp
WHERE deptno<>10
ORDER BY 2, 1;
```



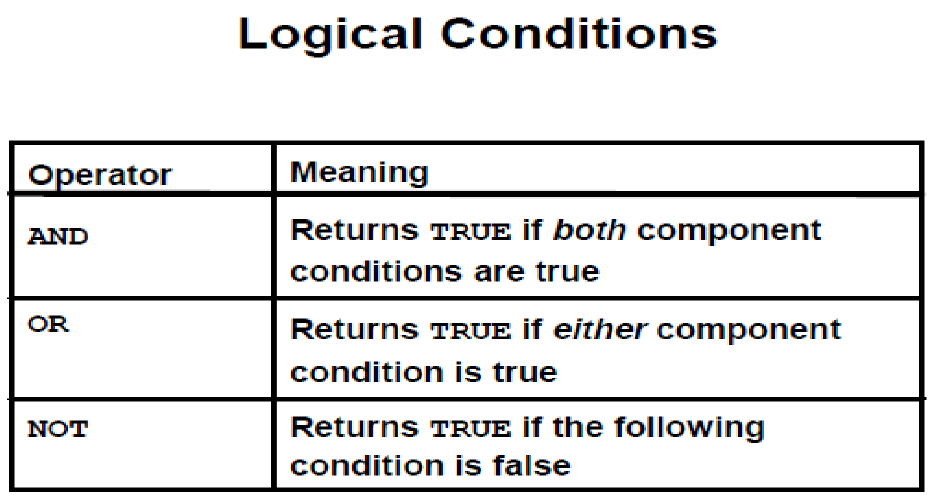
Other Comparison Conditions

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

```
SELECT ELNAME, sal
FROM ap_emp
WHERE sal BETWEEN 4500 AND 9000
ORDER BY sal;
```

```
SELECT ELNAME, deptno
```

```
FROM ap_emp
WHERE deptno IN (10,30)
ORDER BY 2;

SELECT ELNAME, deptno
FROM ap_emp
WHERE ELNAME like 'A%'
ORDER BY 1;

SELECT ELNAME, deptno
FROM ap_emp
WHERE ELNAME like '%N'
ORDER BY 1;

SELECT ELNAME, deptno
FROM ap_emp
WHERE ELNAME like '%A%'
ORDER BY 1;

SELECT ELNAME, sal, comm
FROM ap_emp
WHERE comm IS NULL;

SELECT ELNAME, sal, comm
FROM ap_emp
WHERE comm IS NOT NULL;
```



Logical Conditions

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

```
SELECT ELNAME, sal, deptno
FROM ap_emp
WHERE sal>5000 AND deptno=10;

SELECT ELNAME,sal, deptno
FROM ap_emp
WHERE sal>5000 OR deptno=10;

SELECT ELNAME,sal, deptno, job
FROM ap_emp
WHERE sal<5000 OR deptno=10 AND job='ANALYST';
```

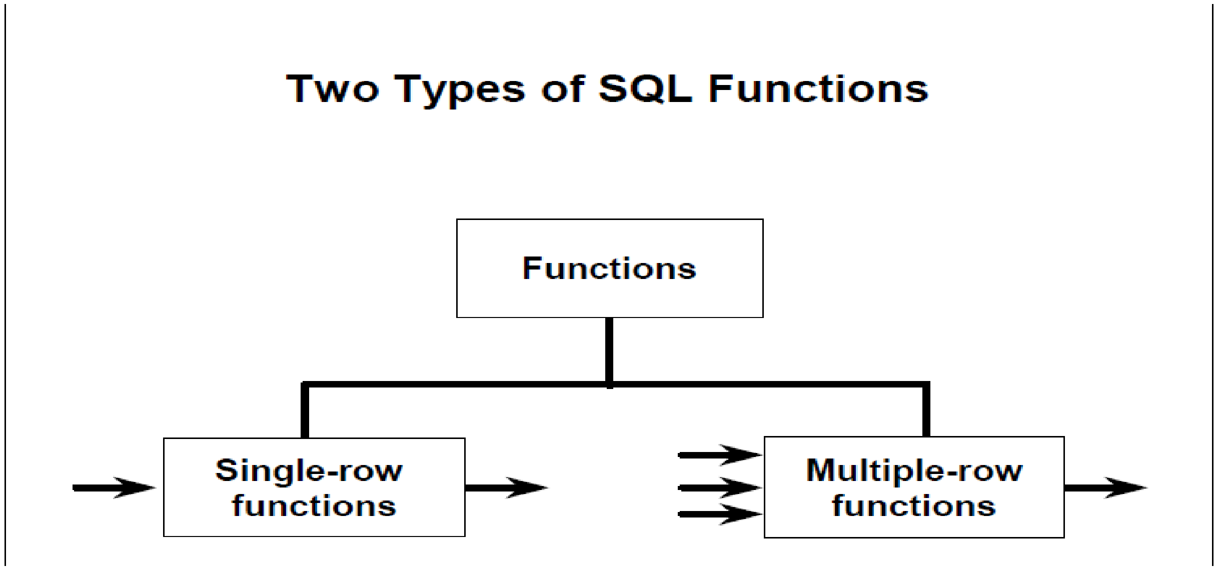
```
SELECT ELNAME,sal, deptno, job
FROM ap_emp
WHERE (sal<5000 OR deptno=10) AND job='ANALYST';

SELECT ELNAME,sal, deptno, job
FROM ap_emp
WHERE sal<5000 OR deptno=10 OR job='ANALYST';

SELECT ELNAME, sal
FROM ap_emp
WHERE sal  NOT BETWEEN 4500 AND 9000
ORDER BY sal;

SELECT ELNAME,deptno
FROM ap_emp
WHERE deptno NOT  IN (10,30)
ORDER BY 2;

SELECT ELNAME, deptno
FROM ap_emp
WHERE ELNAME NOT like 'A%'
ORDER BY 1;
```

SQL Functions

Functions are a very powerful feature of SQL and can be used to do the following:

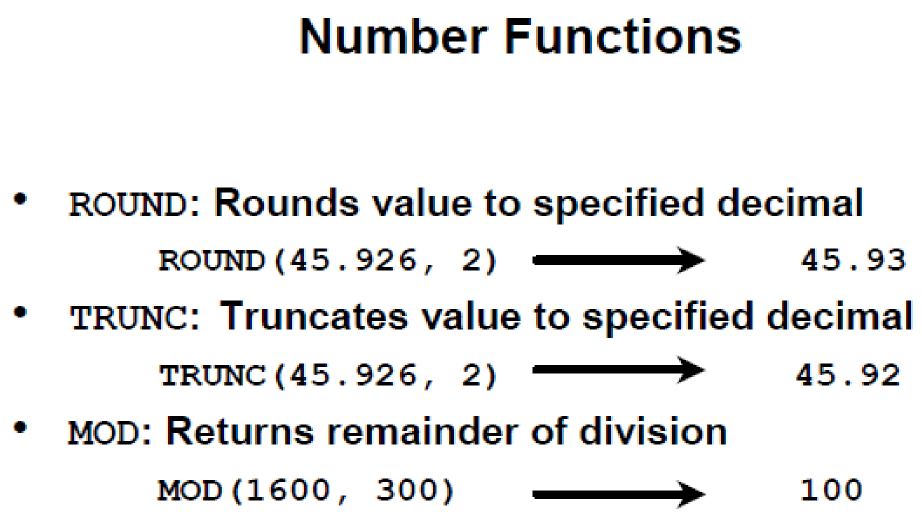
- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

SQL functions sometimes take arguments and always return a value.



ORACLE SQL FUCTIONS
https://docs.oracle.com/database/121/SQLRF/functions.htm#SQLRF006

```
SELECT  ELNAME, lower(ELNAME), substr(ELNAME, 1,2), substr(ELNAME, -3,2), substr(ELNAME, -2),
lower(substr( ELNAME, 1,2))
FROM ap_emp
WHERE UPPER(JOB)='CLERK';

SELECT  ELNAME, LENGTH(ELNAME), sal, LPAD(sal, 10,0)
FROM ap_emp
WHERE UPPER(JOB)='CLERK';
```



Number Functions

- ROUND: Rounds value to specified decimal
  `ROUND(45.926, 2)` $\longrightarrow$ `45.93`
- TRUNC: Truncates value to specified decimal
  `TRUNC(45.926, 2)` $\longrightarrow$ `45.92`
- MOD: Returns remainder of division
  `MOD(1600, 300)` $\longrightarrow$ `100`

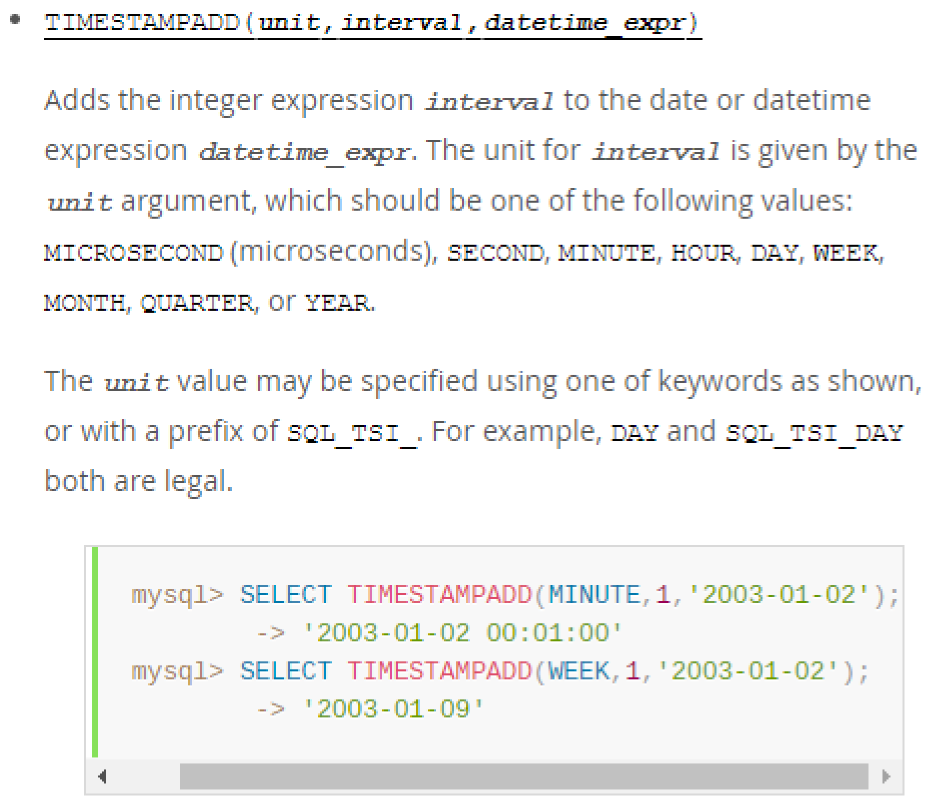
0

```
SELECT ROUND(45.926,2), ROUND( 45.926) FROM dual;
```

```
SELECT TRUNCATE(45.926,2), TRUNCATE( 45.926, 0) FROM dual;
```

```
SELECT ELNAME, sal, MOD(sal, 1000)
FROM ap_emp;
```

DATE ARITHMETIC



- TIMESTAMPADD(*unit*,*interval*,*datetime_expr*)

  Adds the integer expression *interval* to the date or datetime expression *datetime_expr*. The unit for *interval* is given by the *unit* argument, which should be one of the following values: MICROSECOND (microseconds), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

  The *unit* value may be specified using one of keywords as shown, or with a prefix of SQL_TSI_. For example, DAY and SQL_TSI_DAY both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

```
SELECT sysdate
FROM DUAL;

SELECT date_add(sysdate(), interval 14 day)
FROM DUAL;

SELECT SYSDATE()-30
```

```
FROM DUAL;

SELECT ELNAME, hiredate, sysdate()-hiredate "No of days at work"
FROM ap_emp;

SELECT ELNAME, hiredate, (sysdate()-hiredate)/365 "No of years at work"
FROM ap_emp;

SELECT ELNAME, hiredate, date( (sysdate()-hiredate)/365 )  "No of years at work"
FROM ap_emp;


SELECT ELNAME, hiredate, round( (sysdate()-hiredate)/7)  "No of weeks at work"
FROM ap_emp;
```

- `DATE_FORMAT(date,format)`

  Formats the *date* value according to the *format* string.

  The specifiers shown in the following table may be used in the *format* string. The `%` character is required before format specifier characters. The specifiers apply to other functions as well: `STR_TO_DATE()`, `TIME_FORMAT()`, `UNIX_TIMESTAMP()`.

  - `DATE` if the *date* argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH`, and `DAY` parts (that is, no time parts).
  - `DATETIME` if the first argument is a `DATETIME` (or `TIMESTAMP`) value, or if the first argument is a `DATE` and the *unit* value uses `HOURS`, `MINUTES`, or `SECONDS`.

- `DATE_SUB(date,INTERVAL expr unit)`

  See the description for `DATE_ADD()`.

- `DAY(date)`

  `DAY()` is a synonym for `DAYOFMONTH()`.

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, …)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)
`%j`	Day of year (`001`..`366`)
`%k`	Hour (`0`..`23`)
`%l`	Hour (`1`..`12`)
`%M`	Month name (`January`..`December`)
`%m`	Month, numeric (`00`..`12`)
`%p`	`AM` or `PM`
`%r`	Time, 12-hour (*`hh:mm:ss`* followed by `AM` or `PM`)
`%S`	Seconds (`00`..`59`)
`%s`	Seconds (`00`..`59`)
`%T`	Time, 24-hour (*`hh:mm:ss`*)
`%U`	Week (`00`..`53`), where Sunday is the first day of the week; `WEEK()` mode 0
`%u`	Week (`00`..`53`), where Monday is the first day of the week; `WEEK()` mode 1
`%V`	Week (`01`..`53`), where Sunday is the first day of the week; `WEEK()` mode 2; used with **`%X`**
`%v`	Week (`01`..`53`), where Monday is the first day of the week; `WEEK()` mode 3; used with **`%x`**
`%W`	Weekday name (`Sunday`..`Saturday`)
`%w`	Day of the week (`0`=Sunday..`6`=Saturday)
`%X`	Year for the week where Sunday is the first day of the week, numeric, four digits; used with **`%V`**
`%x`	Year for the week, where Monday is the first day of the week, numeric, four digits; used with **`%v`**
`%Y`	Year, numeric, four digits
`%y`	Year, numeric (two digits)
`%%`	A literal `%` character
`%x`	***`x`***, for any "***`x`***" not listed above

```
SELECT date_format(sysdate(), '%d-%b-%y %H:%i:%s ')
FROM dual;

SELECT date_format(sysdate(), '%d-%b-%y %h:%i:%s %p')
FROM dual;

SELECT date_format(sysdate(), '%d-%b-%Y %h:%i:%s %p')
FROM dual;

SELECT date_format(sysdate(), '%d-%bth,%Y hh:%i:%s AM')
FROM dual;

SELECT date_format(sysdate(), '%D-%b “of” %Y %h:%i:%s %p')
FROM dual;

SELECT date_format(sysdate(), '%d')
FROM dual;
SELECT date_format(sysdate(), '%a')
FROM dual;

SELECT date_format(sysdate(), '%b')
FROM dual;

SELECT date_format(sysdate(), '%c')
FROM dual;
```

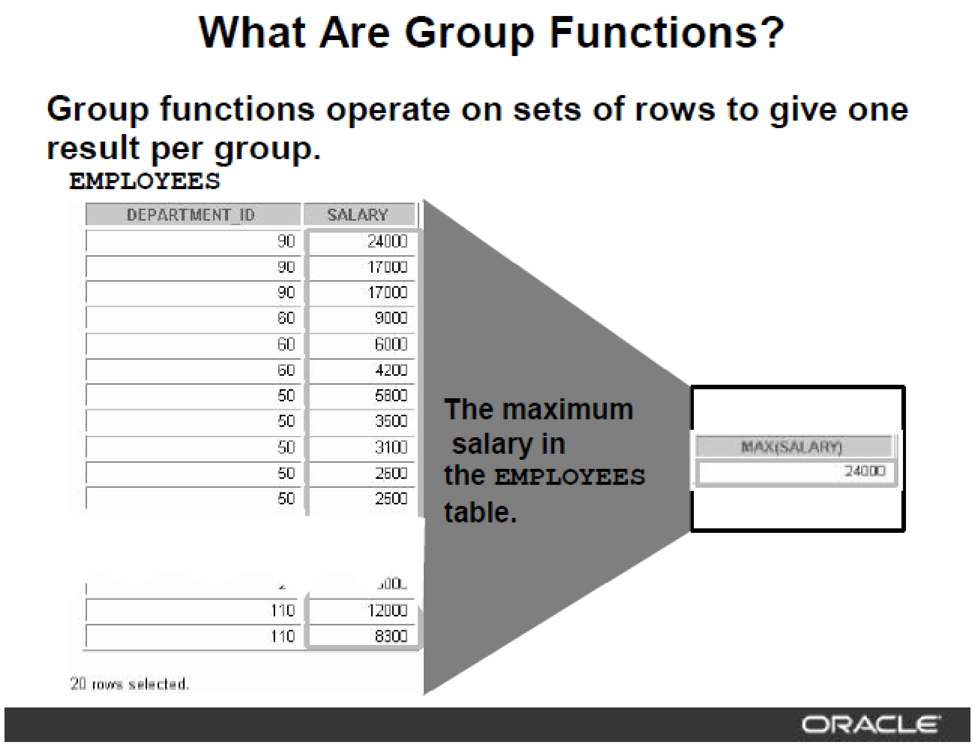
```
SELECT date_format(sysdate(), '%D')
FROM dual;

SELECT date_format(sysdate(), '%e')
FROM dual;

SELECT date_format(sysdate(), '%f')
FROM dual;

SELECT date_format(sysdate(), '%j')
FROM dual;

SELECT date_format(sysdate(), '%k')
FROM dual;

SELECT date_format(sysdate(), '%l')
FROM dual;

SELECT date_format(sysdate(), '%M')
FROM dual;

SELECT date_format(sysdate(), '%m')
FROM dual;

SELECT date_format(sysdate(), '%p')
FROM dual;
```

```
SELECT date_format(sysdate(), '%r')
FROM dual;

SELECT date_format(sysdate(), '%T')
FROM dual;

SELECT date_format(sysdate(), '%U')
FROM dual;

SELECT date_format(sysdate(), '%u')
FROM dual;

SELECT date_format(sysdate(), '%V')
FROM dual;

SELECT date_format(sysdate(), '%v')
FROM dual;

SELECT date_format(sysdate(), '%W')
FROM dual;
```

FORMAT NUMBER TO STRINGS

SELECT sal, CONCAT('$',FORMAT(sal, 0)) from ap_emp;

SELECT sal, CONCAT('$',FORMAT(sal, 2)) from ap_emp;



What Are Group Functions?

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

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
[illegible]	[illegible]
110	12000
110	8300

20 rows selected.

The maximum salary in the **EMPLOYEES** table.

MAX(SALARY)
24000

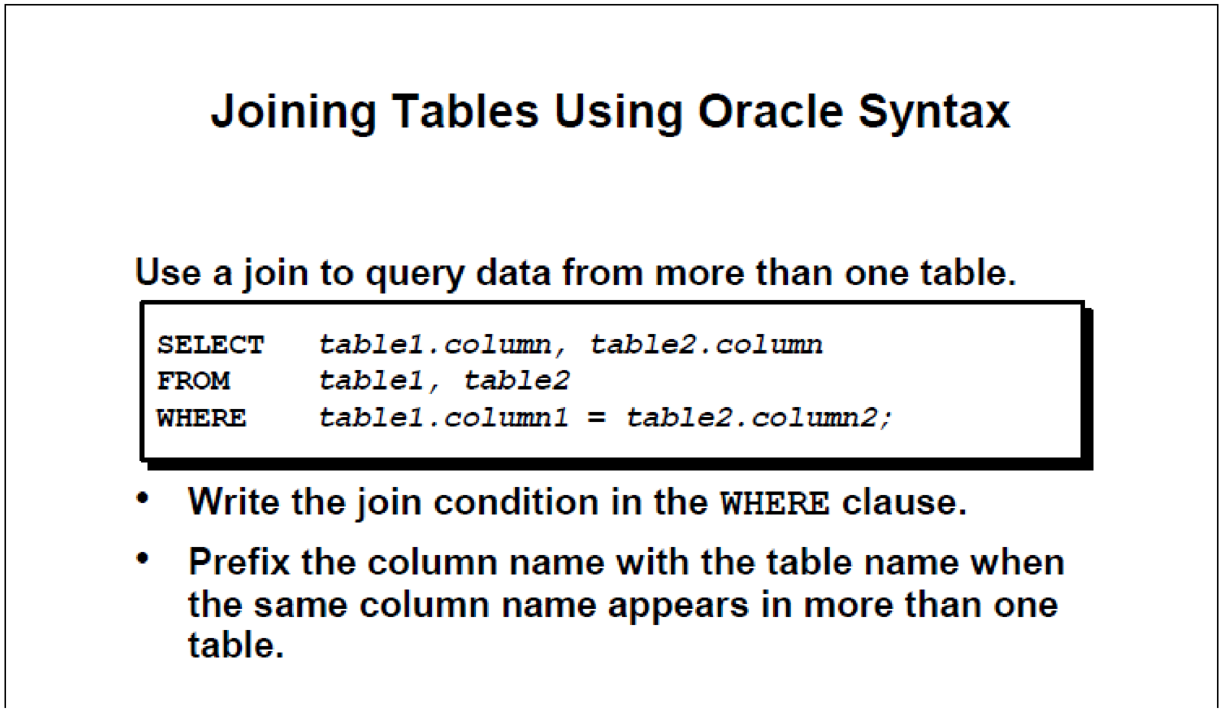
ORACLE

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

```
SELECT COUNT(*), MIN(sal), MAX(sal), AVG(sal), SUM(sal)
FROM ap_emp
WHERE deptno=10;

SELECT MIN(hiredate), MAX(hiredate)
FROM ap_emp
WHERE deptno=20;

SELECT COUNT( DISTINCT job)
FROM ap_emp
WHERE deptno=20;


SELECT AVG(comm)
FROM ap_emp;
```

```
SELECT AVG(IFNULL(comm,0))
from ap_emp;

SELECT deptno, SUM(sal)
FROM ap_emp
GROUP BY deptno
ORDER BY 2;




SELECT deptno, job, AVG(sal)
FROM ap_emp
GROUP BY deptno, job
ORDER BY 1,2;

SELECT deptno, job, AVG(sal)
FROM ap_emp
GROUP BY deptno, job
HAVING AVG(sal)>3000
ORDER BY 1,2;
```



Joining Tables Using Oracle Syntax

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

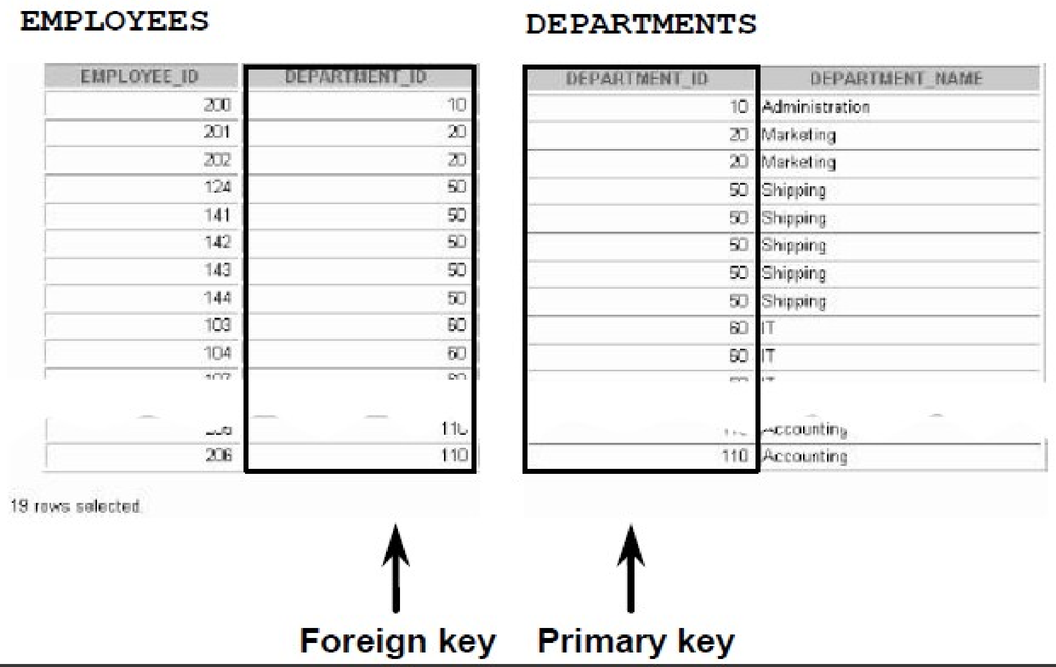
```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column1 = table2.column2;
```

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

Guidelines

- When writing a `SELECT` statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *n* tables together, you need a minimum of `n-1` join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

What Is an Equijoin?


EMPLOYEES
DEPARTMENTS
Foreign key
Primary key

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
205	110
206	110

19 rows selected.

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
110	Accounting
110	Accounting

Copyright © Oracle Corporation, 2001. All rights reserved.

Equijoins

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin*, that is, values in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins* or *inner joins*.

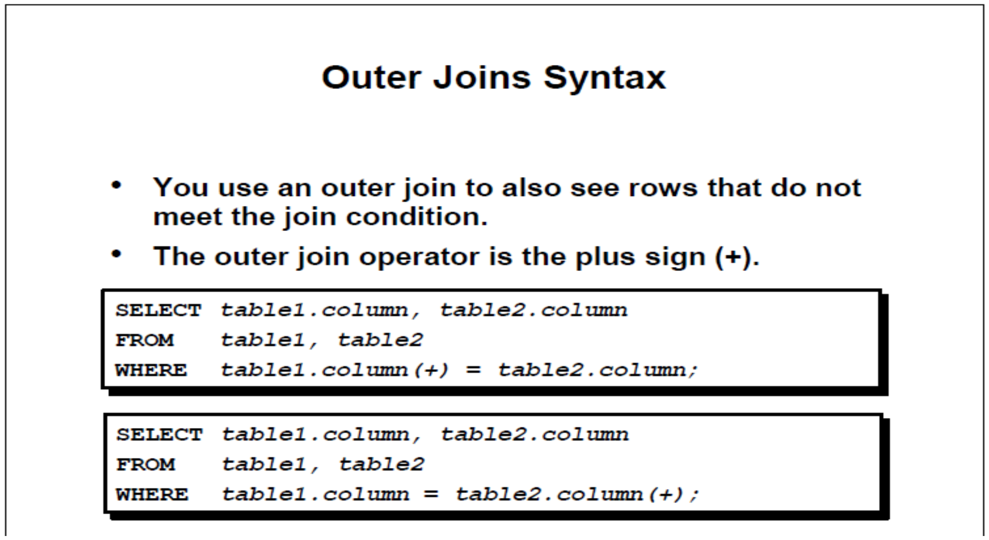
```
SELECT a.empno, a.ELNAME,a.deptno, b.dname
FROM ap_emp a INNER JOIN ap_dept b ON a.deptno=b.deptno
order by 3
;

SELECT a.empno, a.ELNAME,a.deptno, b.dname
FROM ap_emp a JOIN ap_dept b ON a.deptno=b.deptno
order by 3
;

SELECT a.empno, a.projid,b.pname,a.hours
FROM ap_proemp a JOIN ap_project b ON a.projid=b.projid
WHERE a.hours>50
ORDER BY 1
;
SELECT a.empno, a.ELNAME,a.deptno, b.dname,b.loc,c.hours, d.pname
FROM   ap_emp a JOIN ap_dept b ON a.deptno=b.deptno JOIN ap_proemp c ON a.empno=c.empno JOIN
ap_project d ON c.projid=d.projid
WHERE b.loc IN ('NEW YORK', 'DALLAS')  AND c.hours>40
ORDER BY 1;
```

NATURAL JOIN: when FK and PK column of same name, do not need to write join condition.

```
SELECT empno, ELNAME,deptno, dname – column alias on common column will return error.
FROM ap_emp NATURAL JOIN ap_dept;
```

NON-EQUIJOIN: Join based on other than equality operator

```
SELECT a.empno, a.ELNAME,a.sal,b.grade, b.losal, b.hisal
FROM   ap_emp a , ap_salgrade b
WHERE a.sal between b.losal AND b.hisal
ORDER BY b.grade;
```



Outer Joins Syntax

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

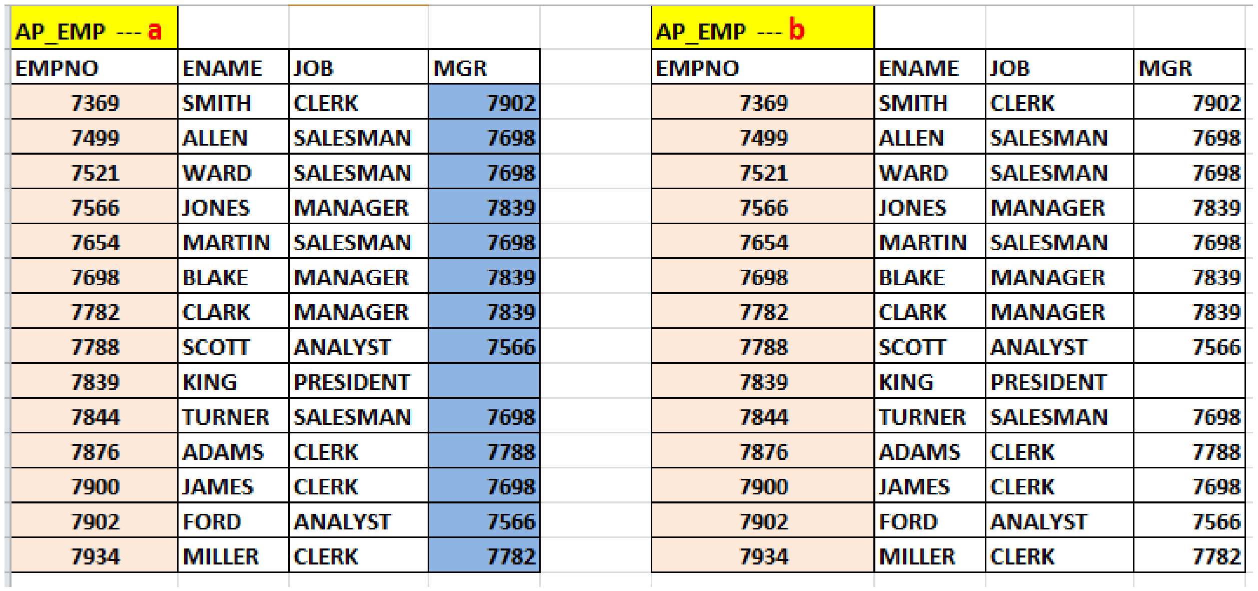
```
SELECT  table1.column, table2.column
FROM    table1, table2
WHERE   table1.column(+) = table2.column;
```

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

```
SELECT a.empno, a.ELNAME,b.deptno,b.dname
FROM   ap_emp a  RIGHT OUTER JOIN  ap_dept b ON a.deptno=b.deptno
ORDER BY a.deptno
;
```

```
SELECT b.empno, b.ELNAME,a.deptno,a.dname
FROM  ap_dept a  LEFT OUTER JOIN ap_emp b ON  a.deptno=b.deptno
ORDER BY b.deptno;
```

SELF JOIN



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

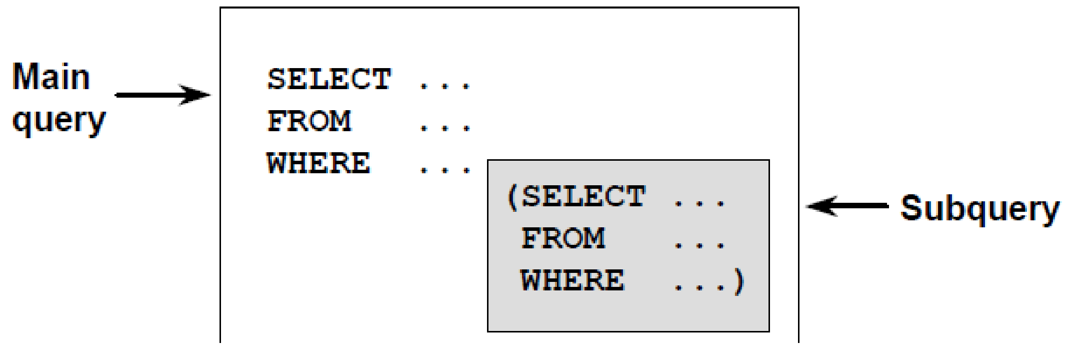
AP_EMP --- b			
EMPNO	ENAME	JOB	MGR
7369	SMITH	CLERK	7902
7499	ALLEN	SALESMAN	7698
7521	WARD	SALESMAN	7698
7566	JONES	MANAGER	7839
7654	MARTIN	SALESMAN	7698
7698	BLAKE	MANAGER	7839
7782	CLARK	MANAGER	7839
7788	SCOTT	ANALYST	7566
7839	KING	PRESIDENT	
7844	TURNER	SALESMAN	7698
7876	ADAMS	CLERK	7788
7900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

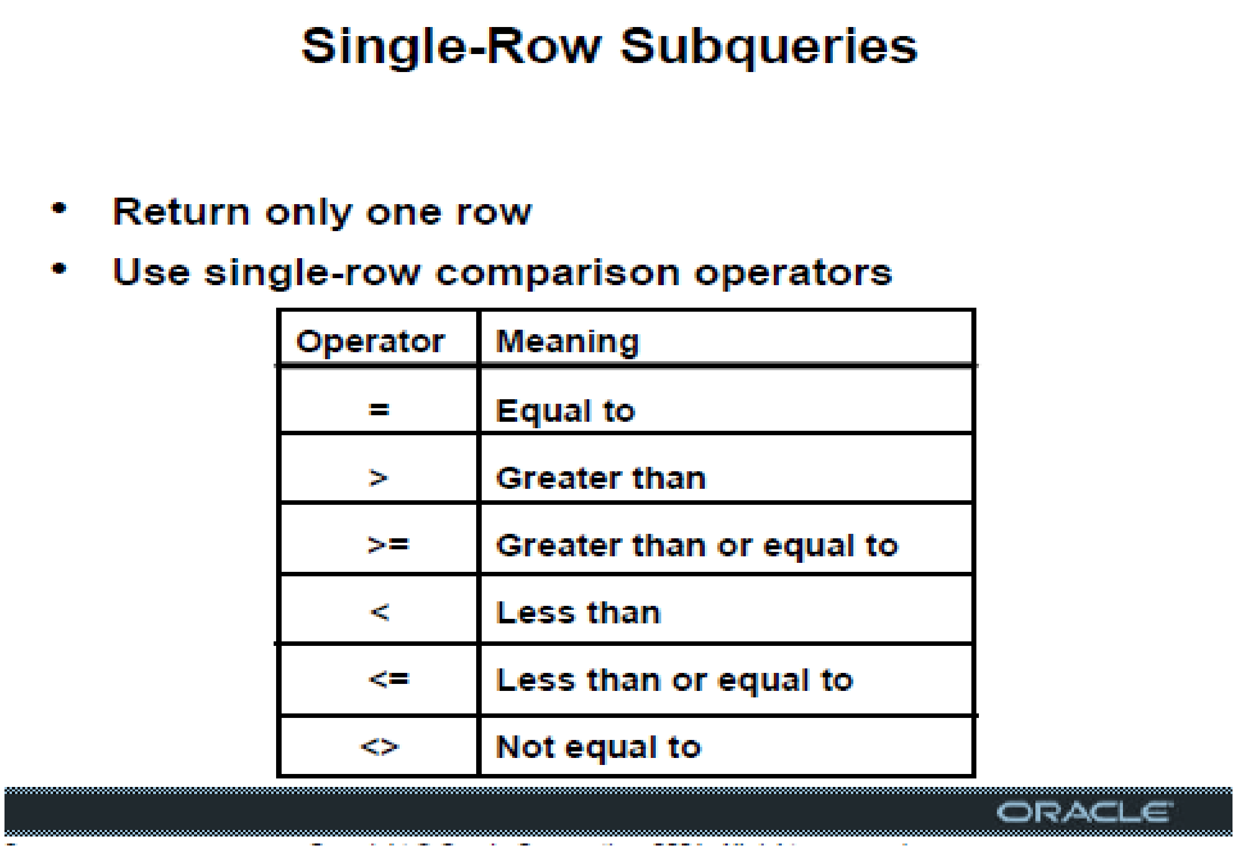
```
SELECT a.empno,a.ELNAME,a.mgr, b.empno " Manager ID", b.ELNAME "Manager Name"
FROM ap_emp a JOIN ap_emp b ON a.mgr=b.empno;
```

USING SUBQUERY

What Is a Subquery?

A subquery is a `SELECT` statement embedded in a clause of another SQL statement.





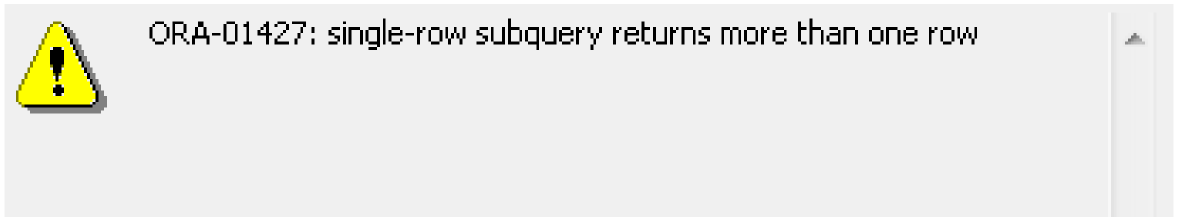
Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

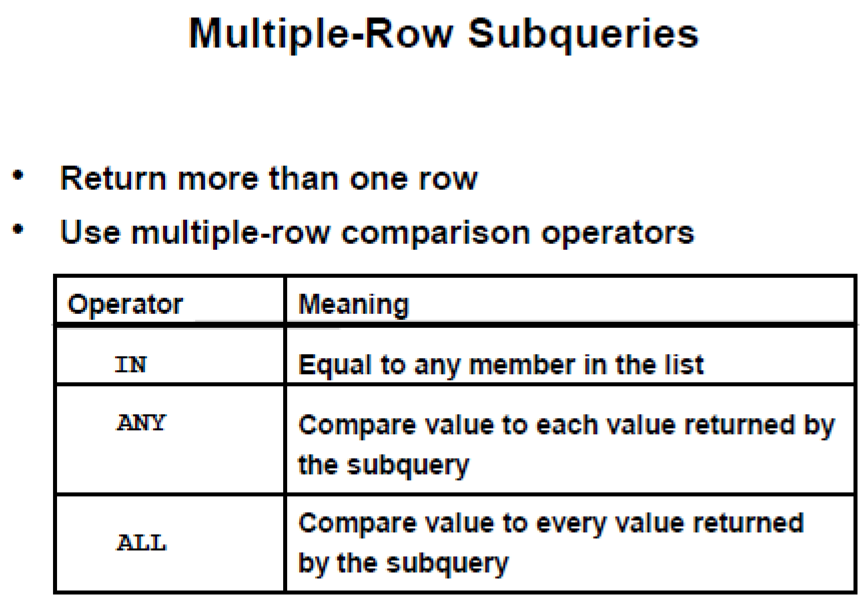
Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

```
SELECT ELNAME, sal
FROM  ap_emp
WHERE sal>( SELECT sal FROM  ap_emp  WHERE ELNAME='ALLEN');

SELECT ELNAME, job
FROM ap_emp
WHERE deptno=(SELECT deptno FROM ap_dept WHERE loc='NEW YORK');
```

```
SELECT ELNAME, job,sal
FROM ap_emp
WHERE  sal>(SELECT  avg(sal) FROM ap_emp);

SELECT ELNAME, job,sal
FROM ap_emp
WHERE  sal>(SELECT  avg(sal) FROM ap_emp) AND
          deptno= (SELECT deptno FROM ap_dept where loc ='NEW YORK')


SELECT job, AVG(sal)
FROM ap_emp
GROUP BY job
HAVING  AVG(sal)=(SELECT MIN(AVG(sal)) FROM ap_emp GROUP BY job);

SELECT ELNAME, sal
FROM ap_emp
WHERE sal > ( SELECT avg(sal) FROM ap_emp GROUP BY deptno);
```



```
ORA-01427: single-row subquery returns more than one row
```



Multiple-Row Subqueries

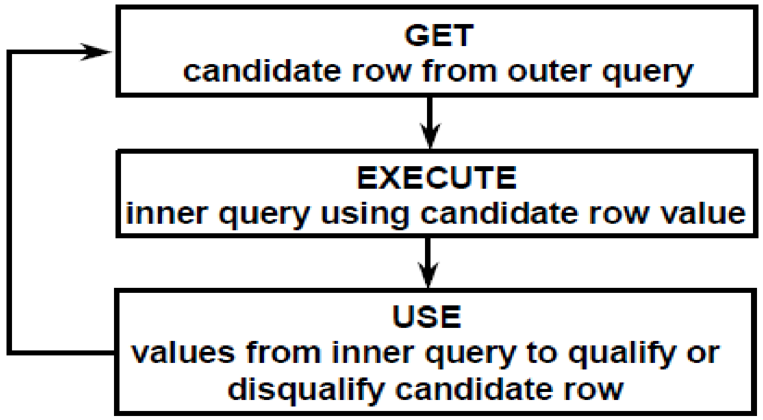
- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

```
SELECT ELNAME, sal
FROM ap_emp
WHERE sal > ALL ( SELECT avg(sal) FROM ap_emp GROUP BY deptno);

SELECT ELNAME, sal
FROM ap_emp
WHERE sal > ANY ( SELECT avg(sal) FROM ap_emp GROUP BY deptno);

SELECT ELNAME, sal
FROM ap_emp
WHERE sal  IN ( SELECT avg(sal) FROM ap_emp GROUP BY deptno);
```

Correlated Subqueries

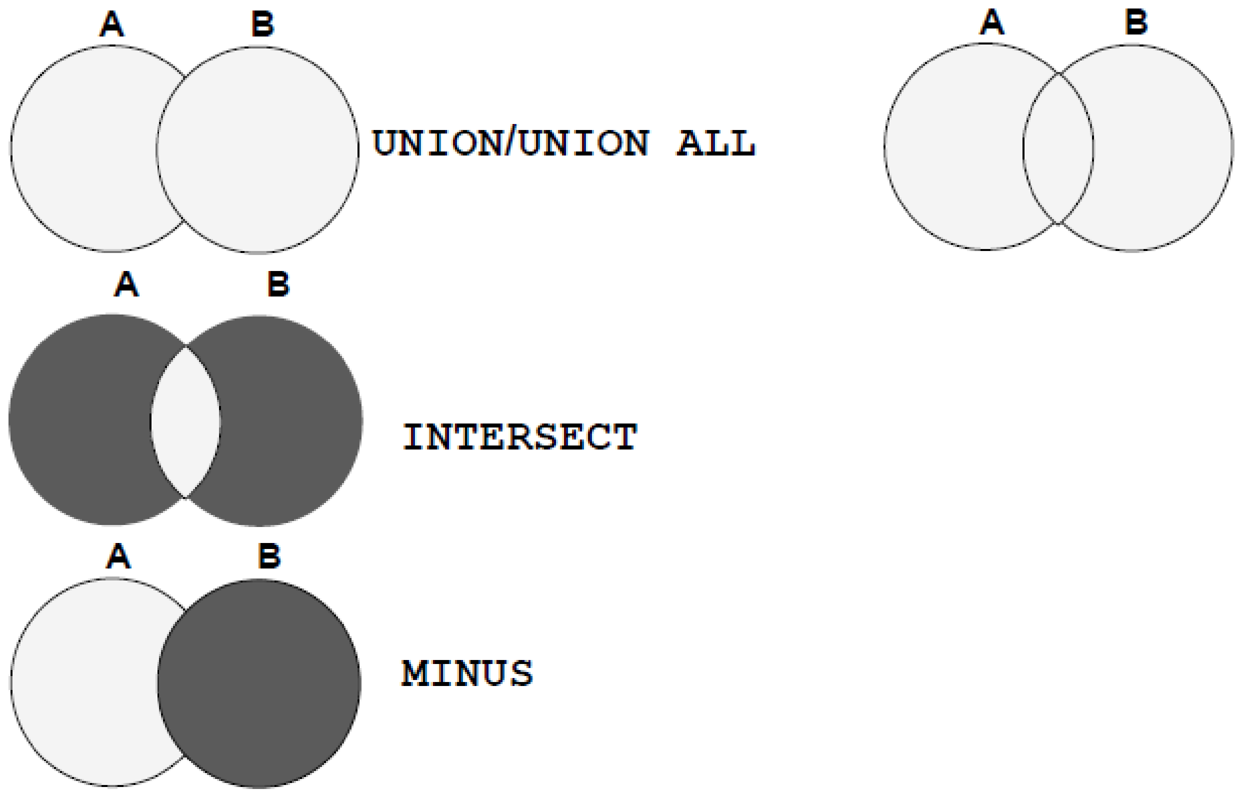
Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.


GET
candidate row from outer query
EXECUTE
inner query using candidate row value
USE
values from inner query to qualify or disqualify candidate row

List employees whose salary is higher than their respective department's average salary

```
SELECT a.empno,a.deptno, a.sal
FROM ap_emp a
WHERE a.sal > (SELECT AVG(sal)  deptavg  FROM ap_emp b WHERE a.deptno=b.deptno);
```

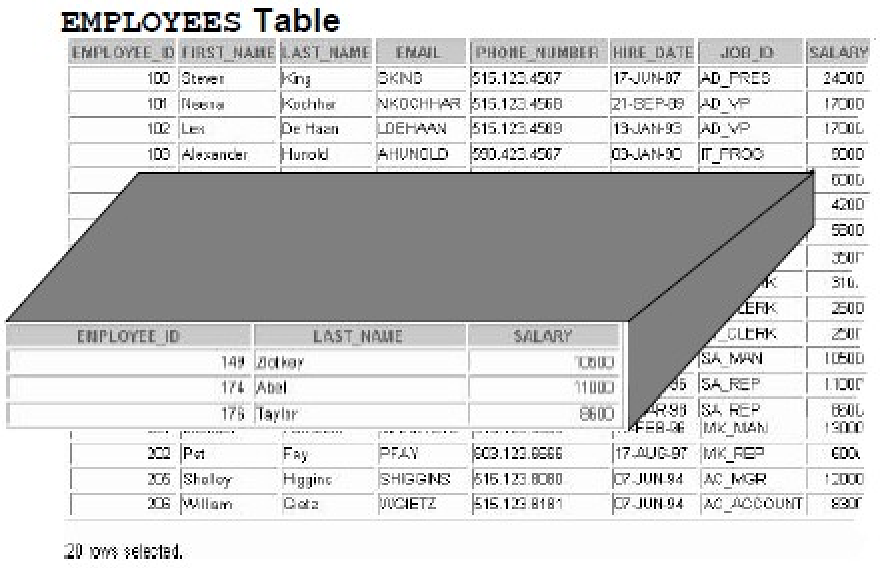
The SET Operators



Operator	Returns
`UNION`	All distinct rows selected by either query
`UNION ALL`	All rows selected by either query, including all duplicates
`INTERSECT`	All distinct rows selected by both queries
`MINUS`	All distinct rows that are selected by the first `SELECT` statement and that are not selected in the second `SELECT` statement

```
SELECT empno, ELNAME, deptno,job
FROM ap_emp
WHERE deptno in (20,30)
UNION
 SELECT empno, ELNAME,deptno, job
FROM ap_emp
WHERE job='ANALYST'
ORDER BY deptno,job;

SELECT empno, ELNAME, deptno,job
FROM ap_emp
WHERE deptno in (20,30)
UNION ALL
 SELECT empno, ELNAME,deptno, job
FROM ap_emp
WHERE job='ANALYST'
ORDER BY deptno,job;

SELECT empno, ELNAME, deptno,job
FROM ap_emp
WHERE deptno in (20,30)
WHERE empno IN
(SELECT empno
FROM ap_emp
WHERE job='ANALYST')
ORDER BY deptno,job;

SELECT empno, ELNAME, deptno,job
FROM ap_emp
WHERE deptno in (20,30)
WHERE empno NOT IN
(SELECT empno
FROM ap_emp
WHERE job='ANALYST')
ORDER BY deptno,job;
```

What Is a View?



EMPLOYEES Table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
							6000
							4200
							5800
							3500
						K	3100
						LERK	2600
						_CLERK	2500
						SA_MAN	10500
					5	SA_REP	11000
					AR-98	SA_REP	8600
					FEB-96	MK_MAN	13000
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

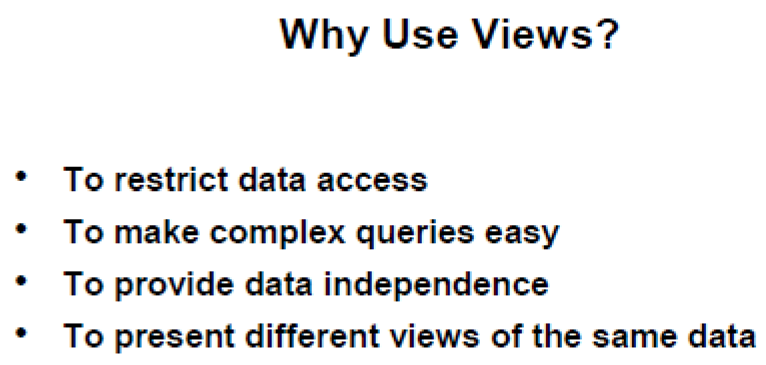
EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.



Why Use Views?

- To restrict data access
- To make complex queries easy
- To provide data independence
- To present different views of the same data

== SIMPLE, SINGLE TABLE VIEW

```
create or replace view testview
as
select * from ap_emp
where deptno=30
;

SELECT * FROM testview;

update testview
set sal=sal+100
where empno=7499;

SELECT * FROM testview;
```

```
update testview
set sal=sal+100
where empno=7499;
```

== VIEW WITH CHECK OPTION

```
create or replace view testview
as
select * from ap_emp
where deptno=30
with  CHECK OPTION;

SELECT * FROM testview;

update testview
set sal=sal+100
where empno=7499;

UPDATE TESTVIEW
SET DEPTNO=20
WHERE EMPNO=7499;
```

== COMPLEX, MULTI TABLES / AGGREGATE functions

```
CREATE OR REPLACE  VIEW dept30_V
AS
SELECT a.empno "Employee Number", a.hiredate "Hire Date", a. sal "Monthly Salary", b.dname "Department Name", b. loc
"Location"
FROM ap_emp a, ap_dept b
WHERE a.deptno=b.deptno AND
        a.deptno=30 ;
```

```
SELECT * FROM dept30_V;

SHOW FULL TABLES
WHERE table_type = 'VIEW' and Tables_in_ap = 'dept30_v';

CREATE OR REPLACE  VIEW emp_proj_detail_V
AS
SELECT a.empno AS Employee, a.ELNAME AS name, a.hiredate AS  HireDate, a. sal AS Salary, b.dname AS Department, c.pname AS
Project, d.hours AS  Hours
FROM ap_emp a, ap_dept b, ap_project c, ap_proemp d
WHERE a.deptno=b.deptno AND
        a.empno=d.empno  AND
        d.projid=c.projid
ORDER BY 4;

SELECT * FROM emp_proj_detail_V;

SELECT name,sum(hours)
FROM ap_emp_proj_detail_V
GROUP BY name
ORDER by 2;

CREATE OR REPLACE  VIEW dept_summary_V
(name,location,minsal,maxsal,avgsal)
AS
SELECT dname , loc, MIN(sal), MAX(sal) , AVG(sal)
FROM ap_emp a, ap_dept b
WHERE a.deptno=b.deptno
GROUP BY dname,loc;

SELECT * FROM dept_summary_V;
```

==== SUBQUERY in FROM CLAUSE

- List employee number, name, department, salary along with their respective department's total number of employee, total salary, average salary, minimum, and maximum salary

```
SELECT a.empno, a.ELNAME, a.deptno,sal, b.deptempcnt , b.depttotsal , trunc(b.deptavgsal) deptavgsal ,
        b.deptminsal, b.deptmaxsal
FROM ap_emp a, ( select deptno, count(*) deptempcnt, sum(sal) depttotsal, avg(sal) deptavgsal, min(sal)
       deptminsal, max(sal) deptmaxsal from ap_emp group by deptno ) b
WHERE a.deptno=b.deptno ;
```

==== WITH CLAUSE [just like a temporary table that can be used multiple times in a query as needed

```
WITH DEPTVAL AS
        ( select deptno, count(*) deptempcnt, sum(sal) depttotsal, avg(sal) deptavgsal, min(sal) deptminsal, max(sal) deptmaxsal
from ap_emp group by deptno)
SELECT a.empno, a.ELNAME, a.deptno,sal, b.deptempcnt , b.depttotsal , trunc(b.deptavgsal) deptavgsal, b.deptminsal,
b.deptmaxsal
FROM ap_emp a, DEPTVAL b
WHERE a.deptno=b.deptno ;
```

== CASE statement [Conditional processing]

Increase employee's salary by $500 if their salary is more than $5000, increase by $300 if their salary is over $3000, otherwise increase by $100.

```
      select sal,  case
                    when sal>5000 then sal+500
                   when sal>3000 then sal+300
                   else
                        sal+100
                   end "Salary Raise"
       from ap_emp;
```

```
update ap_emp
set sal = case
          when sal>5000 then sal+500
          when sal>3000 then sal+300
          else
               sal+100
        end ;
```

== RANK function for TOP-N queries

- List employee number and name of the employee who worked on highest number of projects

```
WITH PROJCNTRANK
AS
( select empno,count(projid) cnt,
        rank () over (order by count(projid) desc) as myrank
        from ap_proemp
        group by empno )
SELECT A.EMPNO, A.ELNAME, B.CNT, B.MYRANK
FROM AP_EMP A JOIN PROJCNTRANK B ON A.EMPNO=B.EMPNO AND
    MYRANK<2;
```

```
-- Subquery in SELECT clause
-- List of department name, and total number of employees in those department
select dname name,
(select count(*)
from ap_emp B
where A.deptno= B.deptno)
as Num_Of_Employees
from ap_dept A ;
```

```
-- Subquery in HAVING clause
-- Department wise Total Salary for those departments which has total salary more than that of department 30
select deptno,sum(sal) Total_Salary
from ap_emp
group by deptno
having sum(sal) >= (select sum(sal) from ap_emp where deptno=30);
```