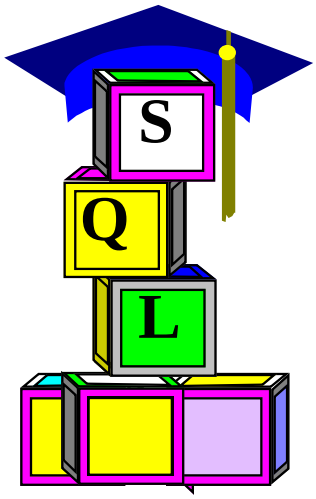
CHAPTER 4

ADVANCED STRUCTURED QUERY LANGUAGE

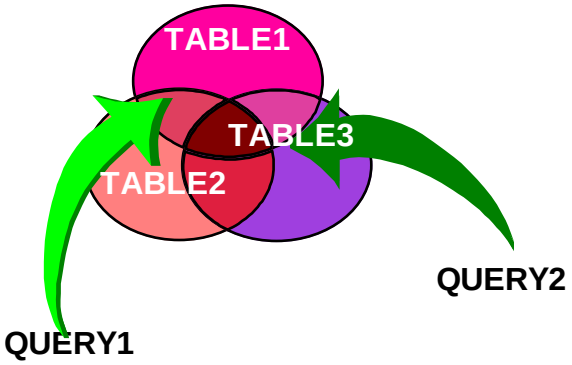


TOPIC OBJECTIVES

This section introduces the more complex SQL statements as used in ORACLE. When you finish this section, you will be able to:

- Create queries called joins which use more than one table
- Comprehend when and how to use subquery statements
- Identify and use the Union, Intersect and Difference commands
- Write SQL Select statements to not only retrieve rows but to insert, delete and update row columns

ADVANCED DML FUNCTIONS



Joins

A join allows the selection of columns from one or more tables based upon comparing two or more columns.

For example, the following figure shows a DEPT table with DEPTNO as a primary key and an EMP table with EMPNO as primary key and DEPTNO as foreign key.

DEPT TABLE

Deptno	Dname
001	SALES
002	MANUF

EMP TABLE

Empno	Mgr	Deptno	Sal	Job
10	40	001	50000	Sales
20	30	002	35800	Tech
30	..	002	43000	Mgr.
40	..	001	70500	Mgr.
50	..	003	34000	Tech

JOIN EXAMPLES (OLD METHOD TO BE DEPRECATED)

A query in which data is retrieved from more than one table. Specifically, select empno, ename, mgr and dname from the appropriate dept and emp tables

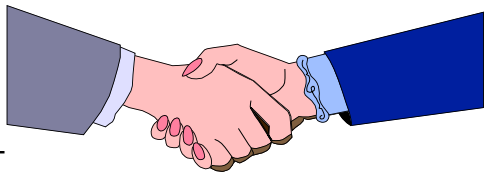
SQL:

SELECT EMPNO, ENAME, MGR, DNAME FROM DEPT,EMP WHERE DEPT.DEPTNO = EMP.DEPTNO ORDER BY EMPNO

RESULT:

EMPNO	ENAME	MGR	DNAME
7782	CLARK	7839	ACCOUNTING
7839	KING		ACCOUNTING
7934	MILLER	7782	ACCOUNTING
7369	SMITH	7982	RESEARCH
7876	ADAMS	7788	RESEARCH
7902	FORD	7566	RESEARCH
7788	SCOTT	7566	RESEARCH
7566	JONES	7839	RESEARCH
....		.....	.....

NOTES ABOUT JOINS



DEPT EMP

In the previous example, DEPT.DEPTNO = EMP.DEPTNO is referred to as the JOIN condition

The two fields in the join condition:

- Must be either both numeric or both character
- Should be the same data type (eliminates conversion overhead)
- May have the same or different column names
- May be compared with equal (=), or with inequalities

Should be used with care.

- They can be quite demanding of the system
- One of the most powerful features of relational systems
- Remember, you must do a join when column information from more than one table is requested by a customer for display purposes.

QUALIFIED NAMES (ON CLAUSE)

Why is the column ambiguous?

- When the columns in the tables to be joined are the same name, you must fully qualify the columns being joined or you will have an ambiguous reference.
- Suppose the following query is needed

List all department managers and their salary

SQL:

SELECT EMP.DEPTNO,ENAME,JOB,SAL,DNAME

FROM DEPT inner joinEMP

ON DEPT.DEPTNO = EMP.DEPTNO

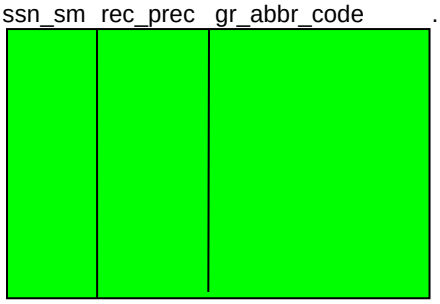
WHERE JOB =‘MANAGER’;

NOTES: Remember that different tables can contain columns with the same name

CORRELATION VARIABLES AND USING CLAUSE

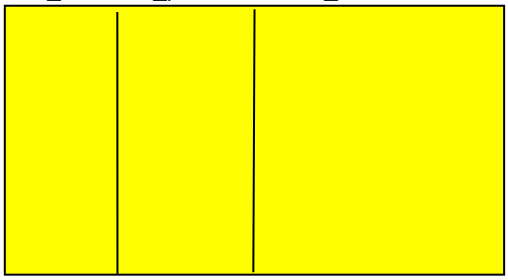
pers_svcmbr_tbl Table

ssn_sm rec_prec gr_abbr_code .....



pers_person_tbl Table

ssn_sm rec_prec name_ind .....



Terminal

Window Edit Options Help

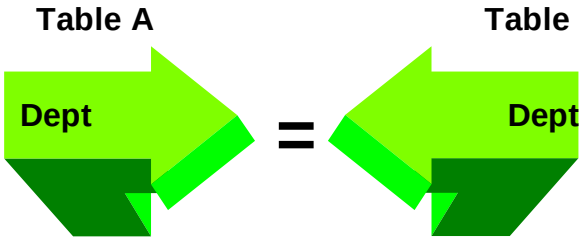
```
SQL> select ps.gr_abbr_code, pp.name_ind
  2  from pers_person_tbl pp INNER JOIN pers_svcmbr_tbl ps
  3  USING (ssn_sm, rec_prec)
```

```
GR_A NAME_IND
---- --------------------
SPCM SIMONDI  VICTORIA
2LTG JONES  DAVID
WO1X ANDERSEN  AARON
SPCM MILLER  VINCENT
PV22 GRECO  VINCENT
PFC3 SHURTLEFF  VINCENT
PFC3 HOUCK  VINCENT
PV11 MCKENZIE  VINCENT
SSG6 MORGAN  VINCENT JR

6899 rows selected.

SQL>
```

Correlation variables are aliases for table names. They are primarily used when columns in different tables have the **same** name. This occurs frequently with primary/foreign key relationships established for referential integrity but may occur in columns with like values as shown below. In this example, we want to identify an applicant who inadvertently has the same id as an actual employee id.

- Also referred to as aliases

ADVANCED DML FUNCTIONS



ORACLE Natural Join

An equi-join is a join where a row in one table has a matching row (based upon equal values) in another table based on one (or more)column(s) in each table.

For example, if you needed to select the social security #, status (rec_prec), name of the individual and state they are in, and unique enlisted status, cumulative retirement points and the unit code from the appropriate tables for soldiers you would write the following:

```
SELECT   ssn_sm,rec_prec, name_ind,states_us,
         mpc,asg_seq_nbr,cum_ret_pts,upc
          FROM pers_person_tbl natural join pers_svcmbr_tlb
```

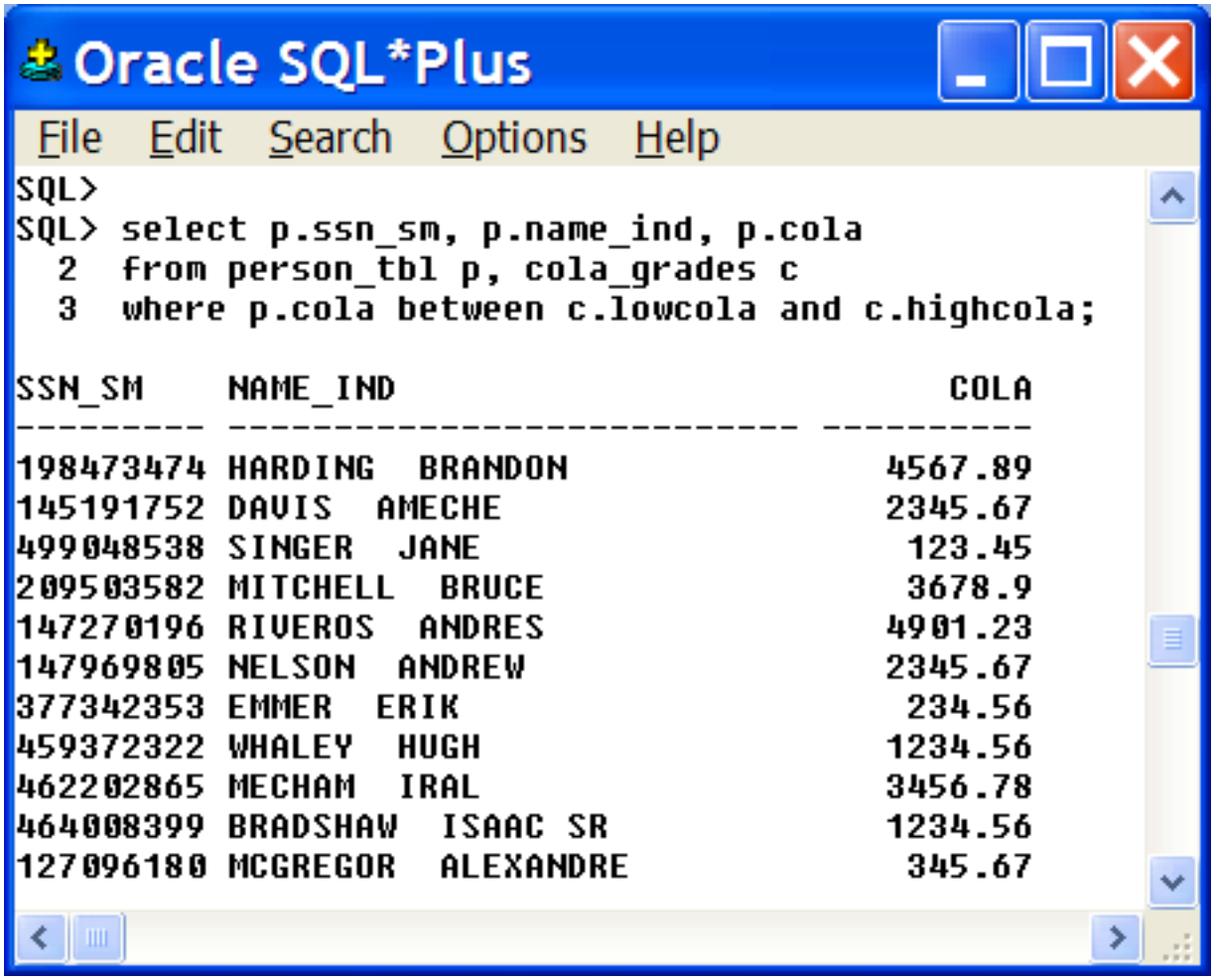
IF AN INDEX IS CREATED ON ssn_sm and rec_prec, THE UNIQUE INDEX CAN BE USED INSTEAD OF SORTING TO RETRIEVE DATA.

Advanced DML Functions

Non-equi-join

A non-equi-join uses operators other than the equal to (=) such as less than (<) and greater than (>). A non-equi-join is a join where a row in one table has a matching row (based upon unequal values) in another table based on one (or more) column(s) in each table.

Example: Find soldiers whose cost of living is between the lowest cost of living allowed and the highest cost of living in the person and cola_grades tables.




```
SQL>
SQL> select p.ssn_sm, p.name_ind, p.cola
  2  from person_tbl p, cola_grades c
  3  where p.cola between c.lowcola and c.highcola;

SSN_SM    NAME_IND                          COLA
--------- ------------------------------ ----------
198473474 HARDING  BRANDON                 4567.89
145191752 DAVIS  AMECHE                    2345.67
499048538 SINGER  JANE                      123.45
209503582 MITCHELL  BRUCE                   3678.9
147270196 RIVEROS  ANDRES                  4901.23
147969805 NELSON  ANDREW                   2345.67
377342353 EMMER  ERIK                       234.56
459372322 WHALEY  HUGH                     1234.56
462202865 MECHAM  IRAL                     3456.78
464008399 BRADSHAW  ISAAC SR               1234.56
127096180 MCGREGOR  ALEXANDRE               345.67
```

.

ADVANCED DML FUNCTIONS

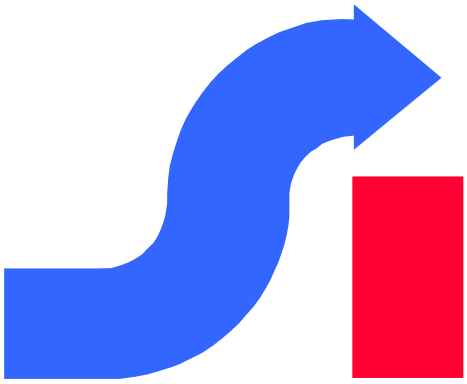


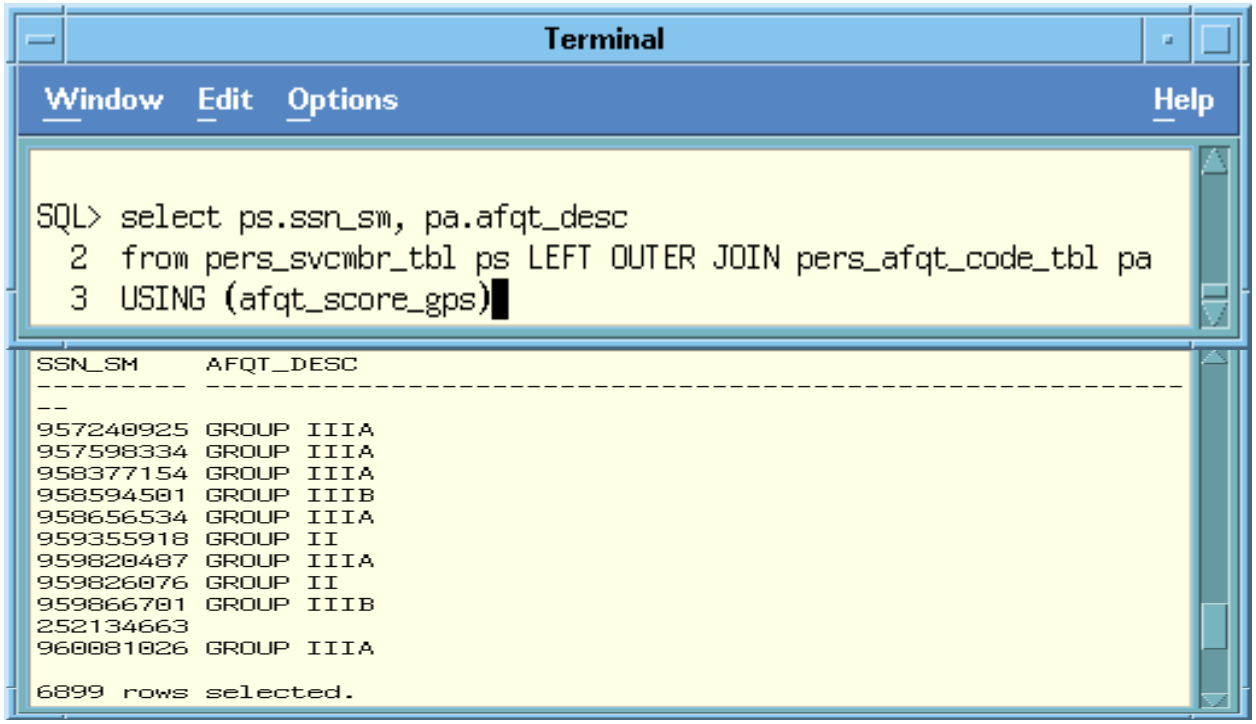
Table 1 Table 2

A column value doesn't exist in the other table

LEFT Outer Join

An outer join may be used to include rows from one table that do not have a matching row in another table.

Example: To find soldiers in the svcmbr table with no physical fitness scores in the pers_afqt_code table table use this outer join:



ANSI/ISO JOIN STATEMENTS FULL OUTER JOIN

Difference between using an "and" versus a "where"

```
SELECT p.name_ind, p.ssn_sm, acum_ret_pt
FROM   pers_person_tbl p FULL OUTER JOIN pers_svcmbr_tbl a
ON     p.ssn_sm = a.ssn_sm and p.rec_prec = a.rec_prec and
p.ssn_sm not in (select ssn_sm from pers_svcmbr_tbl;
```

This particular query will check for all equalities and inequalities in both tables.and bring back all rows.

```
SELECT p.name_ind, p.ssn_sm, cum_ret_pt
FROM   pers_person_tbl p left OUTER JOIN pers_svcmbr_tbl a
ON     p.ssn_sm = a.ssn_sm and p.rec_prec = a.rec_prec
where p.ssn_sm not in (select ssn_sm from pers_svcmbr_tbl)
```

This particular query will check for all inequalities in both tables.and bring back only the inequalities.

PROVIDING HINTS FOR THE OPTIMIZER

In general you should use the cost-based optimization (CBO) approach for all new applications. The cost-based approach generally chooses an execution plan that is as good or better than the plan chosen by the rule-based approach, especially for large queries or multiple joins.

Besides using the ALTER SESSION command to set your optimization plan you can also provide hints within your sql statements.

```
ORACLE>  select   /*+  ALL_ROWS   */  employee_id, salary
            from employee    where department_id = 10;

ORACLE>  select   /*+  FIRST_ROWS   */ employee_id, salary
                FROM employees
                WHERE department_id = 10;
```

Note: You can set the optimizer for all queries by initializing your SESSION OR DATABASE file by putting in the command optimizer_mode = ALL_ROWS or optimizer_mode = FIRST_ROWS

Generally, the hint command is used with the autotrace command to show the I/O and cost of the query when using hints.

SQL> set autotrace on

SQL HINTS

How Hints work? SQL> set autotrace on

```
sql>  SELECT /*+ NO_INDEX */ employee_id, salary, department_id
      FROM employees
      WHERE salary > 1000;
```

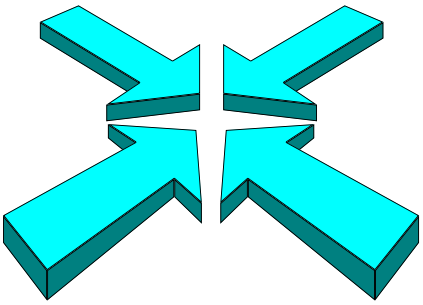
Hints can be used with SELECT, INSERT, UPDATE, and DELETE statements to override the database or session defaults. Hints are defined in comments with the SQL statements. BEWARE: The optimizer may ignore the hints.

- Try to avoid using hints and views or subqueries. Unexpected plans can be generated when hints are used with views or subqueries.
- The Performance Tuning Reference Manual has information about how to used hints with views and subqueries.

The following table lists some of the more common hints that can be used with SQL statements.

RULE	CHOOSE	FIRST_ROWS
ALL_ROWS	FULL	ROWID
CLUSTER	HASH	HASH_AJ
HASH_SJ	INDEX	INDEX_ASC
INDEX_COMBINE	INDEX_JOIN	INDEX_DESC
INDEX_FFS	NO_INDEX	MERGE_AJ
MERGE_SJ	AND_EQUAL	USE_CONCAT
NO_EXPAND	USE_NL	USE_NO_WITH_INDEX

JOIN OPERATIONS



Joining 2 or more table together using NESTED_LOOP JOIN

```
SQL> select /*+ use_nl */ pers_svcmbr_tbl.*, pers_unit_tbl.*
        from pers_svcmbr_tbl, pers_unit_tbl
       where pers_svcmbr_tbl.upc = pers_unit_tbl.upc
```

Here is a look at the different kinds of joins used in Oracle:

Nested Loops Join

A nested loop join is an operation that has two tables, a smaller inner table and an outer table. Oracle compares each row of the inner set with each row of the outer set and returns those rows that satisfy a condition. A nest loop join is commonly seen conditions where an index exists on the inner table. The nested loop join has the fastest response time in many cases (especially with small intermediate result row sets), but the has join often offers the best overall thorough put and faster performance where the intermediate row set is very large.

```
Execution Plan
Plan hash value: 1317748762
| Id  | Operation                    | Name            | Rows  | Bytes |      Cost (%CPU)| Time|
|   0 | SELECT STATEMENT             |                 |  6899 |  2499K|      5537 (1)| 00:01:07 |
|   1 |  NESTED LOOPS                |                 |       |       |
|   2 |   NESTED LOOPS               |                 |  6899 |  2499K|      5537 (1)| 00:01:07 |
|   3 | TABLE ACCESS FULL| PERS_UNIT_TBL 130 15340 |    3 (0)|00:00:01
|*  4 |    INDEX RANGE SCAN          | SVCMBR_IDX6  92 |       1 (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| PERS_SVCMBR_TBL |    53 | 13409 |
73

Predicate Information (identified by operation id):
   4 - access("PERS_SVCMBR_TBL"."UPC"="PERS_UNIT_TBL"."UPC")
Statistics
          0  recursive calls and 0  db block gets
       6232  consistent gets
          0  physical reads and 0  redo size
    1499767  bytes sent via SQL*Net to client
       5569  bytes received via SQL*Net from client
        461  SQL*Net roundtrips to/from client
          0  sorts (memory) and 0  sorts (disk)
       6899  rows processed
```

Hash join

- SQL> select pers_svcmbr_tbl.*, pers_unit_tbl.*
- from pers_svcmbr_tbl, pers_unit_tbl
- where pers_svcmbr_tbl.upc = pers_unit_tbl.upc
- A hash join is an operation that performs a full-table scan on the smaller of the two tables (the driving table) and the builds a hash table in RAM memory. The hash table is then used to retrieve the rows in the larger table. In a hash join, both tables are read via a full-table scan (normally using multi-block reads and parallel query), and the result set is joined in RAM. Is used for equi-joins only.
- Execution Plan
- ----------
- Plan hash value: 3960656270
- | Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time
- | 0 | SELECT STATEMENT | 6899 | 2499K| 74 (2)| 00:0 0:01 |
- |* 1 | HASH JOIN | | 6899 | 2499K| 74 (2)| 00:00:01 |
- | 2 | TABLE ACCESS FULL| PERS_UNIT_TBL | 130 | 15470 3(0)| 00:00:01 |
- | 3 | TABLE ACCESS FULL| PERS_SVCMBR_TBL | 6899 | 1697K| 717(2)| 00:0
-
- Predicate Information (identified by operation id):
- access("PERS_SVCMBR_TBL"."UPC"="PERS_UNIT_TBL"."UPC")
-
- Statistics
- ----------
- 0 recursive calls
- 0 db block gets
- 698 consistent gets
- 0 physical reads
- 0 redo size
- 2253589 bytes sent via SQL*Net to client
- 5568 bytes received via SQL*Net from client
- 461 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 6899 rows processed
-

Sort Merge Join

A sort Merge Join is an operation that retrieves two sets of rows from the target tables and then sorts each row set by the join column. The merge phase combines each row from one set with the matching rows from the other, returning the intersection of the row sets.

```
Execution Plan
Plan hash value: 1618269027
| Id  | Operation          | Name           |
|   0 | SELECT STATEMENT    |               |
|   1 |  MERGE JOIN         |               |
|   2 |   SORT JOIN         |               |
|   3 |    TABLE ACCESS FULL| PERS_UNIT_TBL   |
|*  4 |   SORT JOIN         |               |
|   5 |    TABLE ACCESS FULL| PERS_SVCMBR_TBL |
---------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("PERS_SVCMBR_TBL"."UPC"="PERS_UNIT_TBL"."UPC")
       filter("PERS_SVCMBR_TBL"."UPC"="PERS_UNIT_TBL"."UPC")

Note
-----
   - rule based optimizer used (consider using cbo)


Statistics
----------------------------------------------------------
        213  recursive calls
          0  db block gets
        294  consistent gets
          1  physical reads
          0  redo size
    1508964  bytes sent via SQL*Net to client
       5569  bytes received via SQL*Net from client
        461  SQL*Net roundtrips to/from client
          8  sorts (memory)
          0  sorts (disk)
       6899  rows processed
```

Join Exercises:

USE THE STAFF, ORG, APPLICANT, PERS_PERSON_TBL, PERS_SVCMBR_TBL AND PERS_UNIT_TBL TABLES

1. List ALL employees department numbers and department names with their appropriate division manager's name and his/her salary but only if the employees' salary is greater than 12000 or the employee is from department 10 or the employee is a clerk. See advex71. Use the EMPLOYEE and DEPARTMENT tables.

   (34 rows)

2. For each manager, create a list of people in his/her department whose salary is greater than their manager's salary. See advex72.sql

   RESULTS:

   (10 ROWS RETURNED)

Performing Natural joins

3. Write a natural join on the pers_svcmbr_tbl and the pers_unit_tbl. Save the query as natural1.sql

4. Write a natural join on the jobs table including job_title, employees and the .job_history tables Save the query as natural2.sql

SUBQUERIES

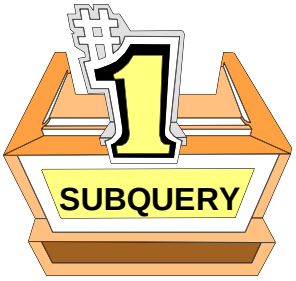
A subquery is a select-from-where expression nested within another select

```
SELECT TEMPID, NAME FROM APPLICANT
 WHERE NAME IN
    (SELECT NAME FROM STAFF
    WHERE JOB='MGR')
```

The subquery produces a value or group of values which complete the main query

- More than one level of subquery nesting can be used
- If a subquery produces no results (rows), you get an empty report

SINGLE VALUED SUBQUERIES



Provides an alternative for the need to use built in functions in the where clause..

TASK: Get the names of all applicants evaluated at the highest education level

<u>SQL:</u>

```
SELECT NAME FROM APPLICANT
 WHERE EDLEVEL =
(SELECT MAX(EDLEVEL)
FROM APPLICANT)
```

RESULTS:

GASPARD
JACOBS

NOTE: If "=" (Or an inequality) precedes the subquery it must be single valued

MULTIVALUED SUBQUERIES

TASK: List the id and name for each employee in the eastern division

SQL:

```
SELECT ID, NAME FROM STAFF
WHERE DEPT IN
        (SELECT DEPTNUMB FROM ORG
        WHERE DIVISION='EASTERN')
ORDER BY ID
```

RESULTS:

All employees from departments 15, 20, and 38

THE ANY PARAMETER

TASK: **List any employee whose salary is greater than at least one department's average salary**

SQL:

```
SELECT NAME FROM STAFF
WHERE SALARY  > ANY
(SELECT AVG(SALARY) FROM STAFF
GROUP BY DEPT)
```

NOTE:

Condition is true if true for at least one value produced by the subquery

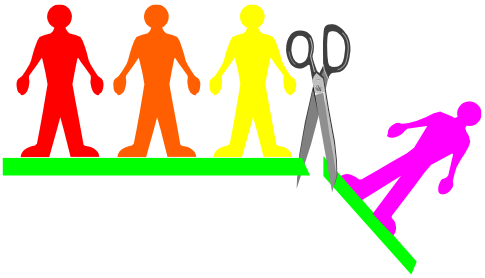
To avoid a multi-scan of a list use the following which finds the smallest value

To do the comparison with:

```
SELECT NAME FROM STAFF
WHERE SALARY  >
(SELECT min(AVG(SALARY)) FROM STAFF
group by dept)
```

Note: Use max(avg(salary)) instead of the ALL parameter. This feature is NOT supported by SQL Server 2005.

DELETING DUPLICATE RECORDS



A common problem that many DBAs and programmers face is trying to purge duplicate rows from a single table. These rows may have been inadvertently (re) imported by the DBA, or mistakenly created by a rogue application program.

The following example shows a particularly efficient way to delete duplicate records from a table. It takes advantage of the fact that a row's ROWID must be unique.

```
DELETE FROM STAFF S
WHERE S.ROWID > (SELECT MIN(X.ROWID)
                 FROM STAFF X
                 WHERE X.ID = S.ID);
```

IN-LINE VIEWS

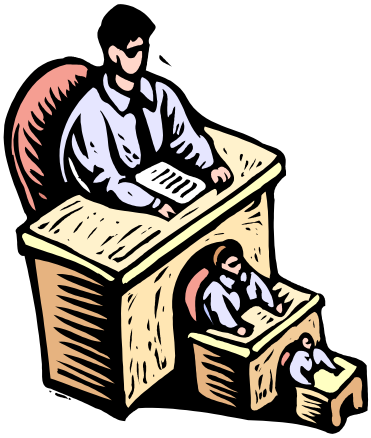


Table View View

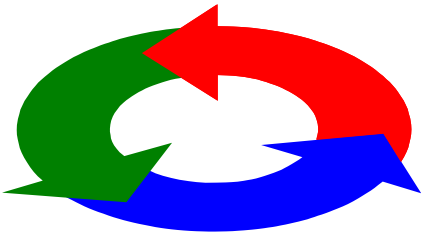
```
SELECT DEPTNO, SUM_SALARIES
   FROM  ( SELECT DEPTNO, SUM(SAL) SUM_SALARIES
                FROM EMP
                  WHERE HIREDATE > TO_DATE('14-Jan-1980')
                     Group by deptno
                        Order by sum(sal) desc
            )
```

In Oracle you can create an in-line view (the subselect above) and it is legal to do an ORDER BV inside a view, including an in-line view as shown above.

Top-N queries are also used with inline views. It uses a virtual column called Rownum. By using an order by in a inline view you can specify only the top 10 rows to be brought back thereby assisting performance. For example:

```
SELECT ename, job, sal, rownum
FROM     (select ename, job, sal from emp
          Order by sal desc)
WHERE   rownum <=10;
```

UTILIZING THE WITH CLAUSE



The ability to reuse a subquery multiple times

The WITH clause allows a developer to define the subquery and then plug the subquery into multiple locations.

```
SQL> WITH AREPEATSUBQRY AS (
        SELECT S1.DEPT, S1.JOB, AVG(SALARY) "AVGSALARY"
        FROM STAFF S1
        WHERE S1.DEPT > 10
        GROUP BY S1.DEPT,S1.JOB)
                SELECT DEPT,JOB,AVGSALARY
                FROM AREPEATSUBQRY
                WHERE DEPT IN
                        (SELECT DEPT
                        FROM AREPEATSUBQRY
                        WHERE  AVGSALARY > 12000)
```

By using this new clause the arepeatsubqry data is calculated just once avoiding an extra scan through a large staff table. It also provides an ease of use mechanism which makes subqueries easier to read.

Exists Parameter - SEMIJOINS

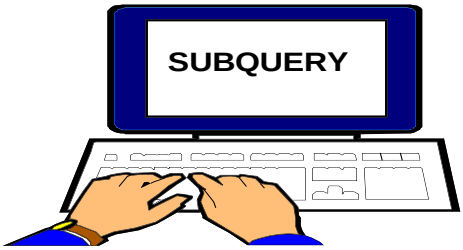
TASK: List each department in Org with at least one employee with more than 8 years of service

SQL:

```
SELECT DEPTNUMB FROM ORG
WHERE EXISTS
(SELECT * FROM STAFF
WHERE DEPT=DEPTNUMB AND YEARS > 8)
```

NOTE:
If subquery produces at least one value, the department is included in the result

EXERCISE 8:



USE THE STAFF, ORG AND APPLICANT TABLES AS NEEDED

1. List the managers in the Midwest division whose salaries are > 20000.

   See ADVEX81.sql

RESULTS:

NAME	DEPT
FRAYE	51

2. List the managers whose department's average salary is greater than the overall average salary. See advex82.sql or inline.sql for a more advanced feature to be covered in the next section.

RESULTS:

DEPT	NAME	SALARY	JOB
10	MOLINARE	22059.2	MGR
10	LU	20010	MGR
10	DANIELS	19260.3	MGR
10	JONES	21234	MGR

…… (more than 4 rows)

3. Find the employee with the second highest salary. See advex83.sql

   RESULTS:

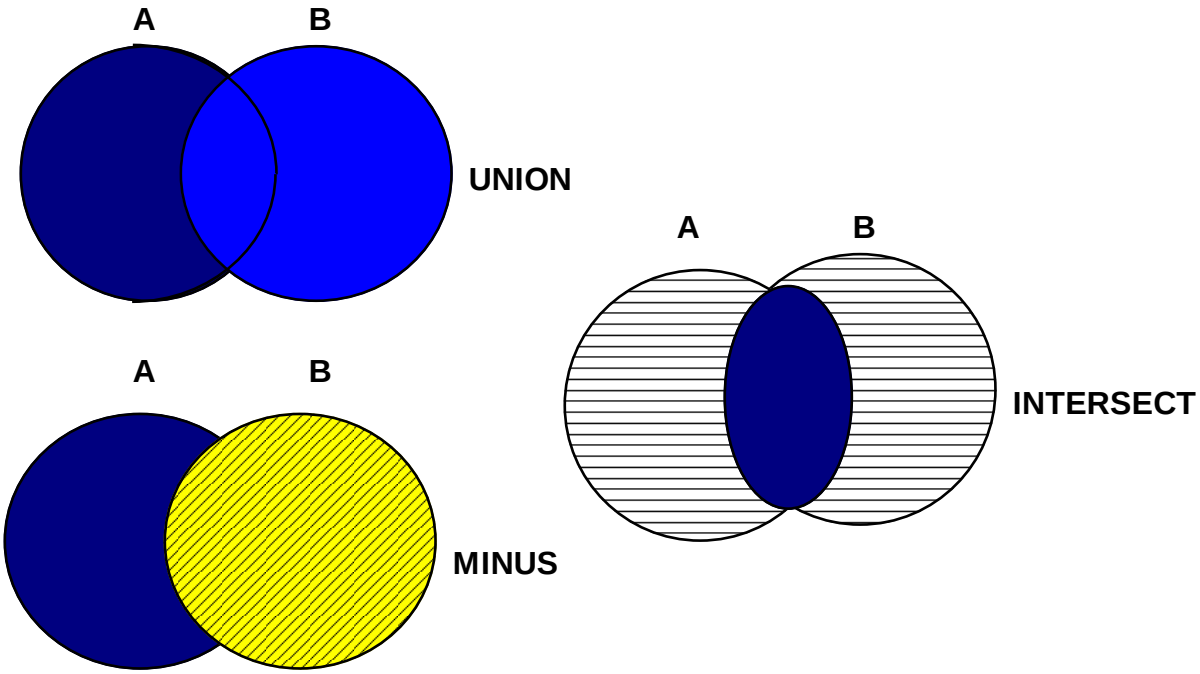
NAME	SALARY

   (1 row)

4. Retrieve the employee number, name, salary, and commission of all employees who earn more than $20,000.00 only if any employee earns less than $350.00 commission. (Hint: use an exist statement). See advex84.sql

   RESULTS: 7 rows

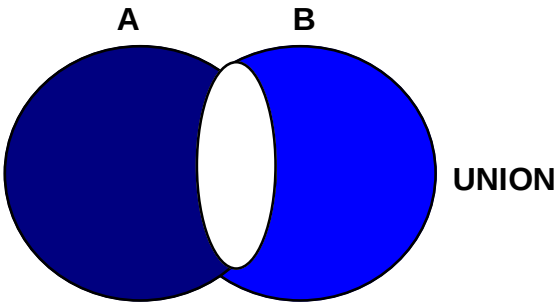
SET OPERATORS (Conjunctions)



Set operators combine the results of multiple queries into a single result. The following table lists SQL set operators in order of precedence from high to low.

Operator	Function	Example
UNION	Combines queries to return all distinct rows returned by any individual query (OR)	...SELECT .. UNION SELECT ...
INTERSECT	Combines queries to return all distinct rows returned by every individual query. (AND)	...SELECT.... INTERSECT SELECT..
MINUS	Combines queries to return all distinct rows returned by the first query but not the second query (AND NOT)	....SELECT.... MINUS SELECT.....

UNION



A **UNION** conjunction would combine rows in all queries in the union eliminating duplicate rows.

- The results would be similar to those returned when using a WHERE clause with an OR logical operator.

TASK: List employees by department. Designate for each employee if his years of service are from 0 to 5, from 6 to 10, or greater than 10

SQL:

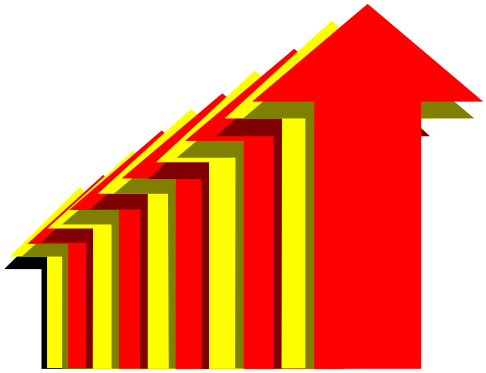
```
SELECT DEPT, ID, NAME, '0-5 YEARS'
FROM STAFF
WHERE YEARS < 6
UNION
SELECT DEPT, ID, NAME ,'6-10 YEARS'
FROM STAFF
WHERE YEARS BETWEEN 6 AND 10
UNION
SELECT DEPT, ID, NAME, 'MORE THAN 10 YEARS'
FROM STAFF
WHERE YEARS >10
ORDER BY 1,2
```

RESULT:

DEPT	ID	NAME	0 - 5 YEARS
10	160	MOLINARE	6 - 10 YEARS
10	210	LU	6 - 10 YEARS
10	240	DANIELS	0 - 5 YEARS
10	260	JONES	MORE THAN 10 YRS

.. ..

CONSIDER IN OR UNION IN PLACE OF OR



1 2 3 4 5 6 7 8 9 10 …. SQL statements are merged together

In general, always consider the IN and/or UNION verb instead of the OR verb in WHERE clauses. Using the OR predicate on an indexed column causes the optimizer to perform a full table scan rather than an indexed retrieval.

In the following examples, both division and deptnumb are indexed. Consider the following:

```
SELECT DEPTNAME,LOCATION
    FROM ORG
    WHERE DEPTNUMB = 10
    OR DIVISION = 'EASTERN';
```

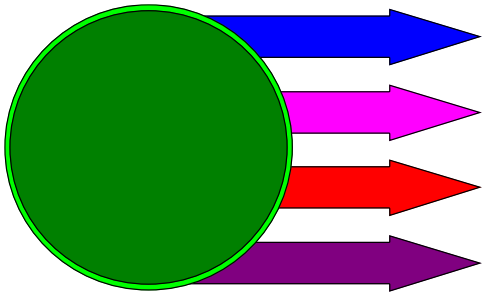
```
SELECT DEPTNAME,LOCATION
        FROM ORG
        WHERE DEPTNUMB = 10
OR      DEPTNUMB = 42
OR      DEPTNUMB = 84;
```

To improve performance, replace this code with:

```
SELECT DEPTNAME,LOCATION
    FROM ORG
    WHERE DEPTNUMB = 10
    UNION ALL
SELECT DEPTNAME,LOCATION
    FROM ORG WHERE
    DIVISION = 'EASTERN';
```

```
SELECT DEPTNAME,LOCATION
        FROM ORG
WHERE DEPTNUMB IN(10,42,84)
```

CONSIDER UNION ALL IN PLACE OF UNION



Union ALL does not eliminate duplicate rows

Programmers of complex query statements that include a UNION clause should always ask whether a UNION ALL would suffice. The UNION clause forces all rows returned by each portion of the UNION to be sorted and merged and duplicates to be filtered before the first row is returned to the calling module. A UNION ALL simply returns all rows including duplicates and does not have to perform any sort, merge, or filter. Most statements that do include a UNION clause can in fact replace it with a UNION ALL. Consider the following example:

```
Select  DEPTNO,ENAME,JOB, HIREDATE,'STAFF MEMBER'
FROM EMP
WHERE HIREDATE  BETWEEN '11-DEC-88' AND '11-DEC-99'
UNION
SELECT 11,NAME,COMMENTS,SYSDATE, ' APPLICANTS'
FROM APPLICANT
```

To improve performance, replace this code with:

```
Select  DEPTNO,ENAME,JOB, HIREDATE,'STAFF MEMBER'
FROM EMP
WHERE  HIREDATE  BETWEEN '11-DEC-88' AND '11-DEC-99'
UNION ALL
SELECT 11,NAME,COMMENTS,SYSDATE, ' APPLICANTS'
FROM APPLICANT
```

Exercise 9 on Union:

1. List all the applicants and staff members and employees, order by employee ID. Label each person as either "STAFF" ”EMPLOYEE ”or "APPLICANT”. Display the employee id, names and job classifications See ADVEX91.sql Use the STAFF, EMP and APPLICANT tables for this query

   RESULTS:

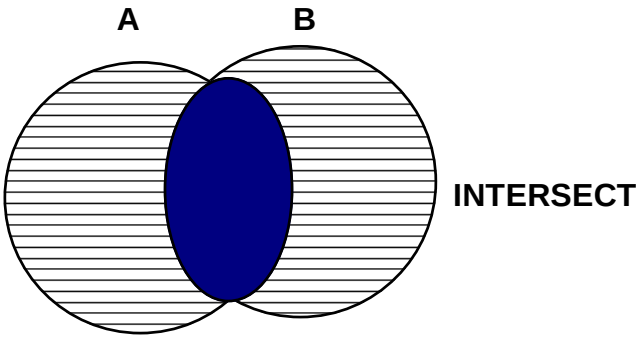
   59 rows selected

2. Provide an employee list, which indicates which employees make less than the average salary and which employees have salaries greater than, or equal to the average salary. See EX92.sql. Use the STAFF tables.

   RESULTS:

   35 rows selected

SET OPERATOR INTERSECT



The INTERSECT clause results would be similar to those returned when using a WHERE clause with an AND logical operator.

The following example would produce an empty result table because an employee is only assigned one DEPTNO at any given time

SELECT EMPNO, DEPTNO FROM EMP WHERE DEPTNO = 001

INTERSECT

SELECT EMPNO, DEPTNO FROM EMP WHERE DEPTNO = 003

The following use of intersect would give a non-empty results table:

SELECT EMPNO FROM EMP

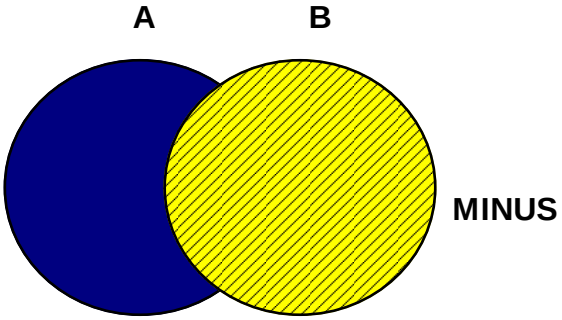
INTERSECT

SELECT EMPNO FROM PROJECTB

Results:

EMPNO
10
20

THE MINUS CONJUNCTION



The minus conjunction is similar to using the 'AND NOT' logical operator is a WHERE clause of a SELECT statement

SELECT EMPNO, SAL FROM EMP
WHERE SAL > 46000
MINUS
SELECT EMPNO, SAL FROM EMP
WHERE DEPTNO = 001

Results:

EMPNO	SAL
50	90000

SELECT EMPNO CONTRACTOR FROM PROJECTB
MINUS
SELECT EMPNO FROM EMP

Results:

CONTRACTOR
578
789

MODIFY FUNCTIONS

SQL Insert Selected Columns

```
INSERT INTO ORG(DEPTNUMB, DEPTNAME, LOCATION)
VALUES (95,'COASTAL', 'HOUSTON')
```

- Columns not specified are given the value null
- A value must be specified for any column defined as not null in the table create statement

```
INSERT INTO MYSTAFF (ID, NAME, DEPT)
SELECT ID, NAME,DEPT
FROM STAFF WHERE JOB = 'CLERK'
```

Copies all clerk rows from STAFF to table MYSTAFF

The insert (as with all updates) may be performed on a table if

- You created the table
- You have been granted insert authority on the table

MULTITABLE INSERT STATEMENT

Insert into 2 tables with 1 statement

The multitable INSERT statement:

- Allows records to be inserted into multiples at the same time.
- Can be used to copy data from one or more tables to a different set of target tables
- Supports two types of operations
    1. Conditional
    2. Unconditional

The values clause can only contain columns referenced in the SELECT statement in the subquery.

```
INSERT ALL  INTO T1 [VALUES(…..) \
            INTO t2 [VALUES(….) ]
   ……
SELECT ……
```

EXAMPLE OF UNCONDITIONAL INSERT

```
INSERT ALL
     INTO product_activity VALUES(today, product_id,
     quantity)
     INTO product_sales VALUES(today, product_id, total)
   SELECT trunc(to_date(to_char(order_date,'dd-MON-yy')))
       today, product_id, SUM(unit_price) total,
       SUM(quantity) quantity
FROM orders, order_items
   WHERE orders.order_id = order_items.order_id AND
      trunc(to_date(to_char(order_date,'dd-MON-yy')))=
      TRUNC(SYSDATE)
GROUP BY trunc(to_date(to_char(order_date,'dd-MON-yy'))),
         product_id;
```

tables as part of a single DML statement

- Can be used in data warehousing systems to transfer data from one or more operational sources to a set of target tables
- Is used internally for refreshing materialized views
- Allows you to still benefit from:
  - Parallelization
  - Direct-load mecanism

EXAMPLE OF A CONDITIONAL ALL INSERT

```
INSERT ALL
  WHEN product_id IN (SELECT product_id
                      FROM   promotional_items)
THEN
    INTO promotional_sales
    VALUES(product_id,list_price)
  WHEN order_mode = 'online'
THEN
    INTO web_orders VALUES(product_id,
    order_total)
   SELECT product_id, list_price, order_total,
order_mode
FROM   orders;
```

INSERT {ALL | FIRST |

WHEN c1 then INTO t1 [VALUES (…)]

WHEN c2 then INTO t2 [VALUES (…)]

…..

[ELSE INTO tn [VALUES (…)]]

SELECT …..

The INSERT FIRST syntax will execute the first INTO clause that matches on the expression. It will, then, skip all following WHEN clauses.

RESTRICTIONS and USES FOR MULTITABLE INSERTS

The following are restrictions with multitable inserts.

A multitable insert cannot:

- Be used with views and materialized. Only tables are supported.
- Be performed on a remote table
- Use a collection expression
- Be parallelized in a Real Application Cluster environment.
- Use plan stability.
- Use a sequence in a subquery.

Note: The multitable INSERT statement offers benefits in DSS and large batch processing environments. Multitable INSERTS can be used with:

- Parallel operations
- Direct load operations
- Refreshing materialized views.

Update a single row

Task: Update employee number 150 to change job description to Manager, Department to 95 and provide a 15% increase in salary.

SQL:

```
UPDATE STAFF
SET JOB='MGR', DEPT=95,  SALARY=SALARY * 1.15
WHERE ID=150
```

This example updates the department, job and salary for ID 150

Update multiple rows

SQL:

```
UPDATE STAFF
    SET COMM = COMM + 500
    WHERE DEPT = 38
```

This query gives every employee in Department 38 a $500 commission bonus

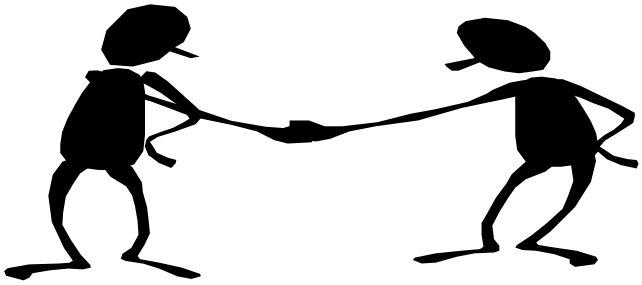
COMPLEX UPDATE

Task: Provide all employees who worked for Project number 50 a bonus equal to 15% of their salary. This requires that you search the Project table to find all employees in Project 50 and then update the employees Bonus field in the Staff table

```
Update Staff
Set Bonus =  salary * .15
where id in
(Select empno
 from project
where projectno = 50);
```

Result: All employees attached to Project number 50 are given a 15% bonus.

MERGING DATA IN ORACLE TABLES



With one statement we can either insert or update

The Oracle12C merge command allows you to decide based upon a condition whether you will be inserting or updating. Implicit in the use of the merge command is the concept that you are merging the contents of one table into another based on whether the values exist in the second table, extending the table join principles reviewed earlier. For example, assume that a transaction file being read in PLSQL has both new employees salaries and old employees salaries that require updating. New employees are not immediately assigned department numbers. So we can use the MERGE command to decipher whether we are going to insert new employees or simply update them.

```
MERGE into staff s1
Using org o2 on (s1.dept = o2.deptnumb )
        When matched then update set s1.salary = :new_salary
        When not matched then insert (id,name,dept,years,job,salary,comm.)
        Values (:new_id,:new_name,:new_dept,:new_years,
                :new_salary,:new_comm.);
```

The above example states in line 1 “Merge rows in the STAFF table”. The second line specifies “to match the data using the Org and Staff tables). The third and fourth lines define that if a employee has a department number in both tables then his/her salary is updated otherwise the NOT MATCHED says they don’t exist and can be inserted.

NOTE: Starting In 12C you can insert, update or delete with the MERGE command.

COMPLEX DELETE

Task: Delete all employees who worked for Project number 50.

```
Delete from  Staff
where id in
(Select empno
 from project
where projectno = 50);
```

Result: All employees attached to Project number 50 are deleted

LAB 4 - SELECTING ROWS FROM MORE THAN ONE TABLE

1. TASK: Retrieve all employees in each department who have the highest salaries. Print out the department name along with their id, name, and salary. Use the STAFF and ORG tables. See Lab41.sql

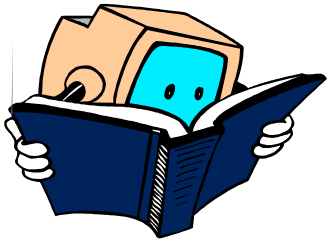
2. TASK: Select the employee number, name, job, salary, department and department name for all employees whose total earnings is in the range of 15000 to 30000 dollars and are salesman and were hired before 1999. Sort the data by department name and job. Use the **emp** AND **dept** tables for the query. See lab42.sql (Hint: The column SAL is monthly salary) and assume the commission is current and usable as is.)

3. TASK: Display the rows in **emp** table that do not have a match in the **dept** table (an employee who does not have a department number in the dept table) as well as those employees with department numbers that do have a match in the dept table.. Optional - See if you can just return an emp row(s) that does not have a match in the dept table (This will require you to add your name and relevant data into the emp table. Use department number 11 for your department See lab43.sql

.

.

LAB 4 - SELECTING ROWS FROM MORE THAN ONE TABLE



4. a. Insert your name and relevant information into the ***applicant*** table.

   b. Insert a row into the ***org*** table that creates a new Division (Southwest), department number (77), department name (Dallas), location (Plano), and manager (10).

   c. The company has hired you. Move your applicant information from the ***applicant*** table to the ***staff*** table. Your salary is $50,000.00, you have no commission, and your job status is “MGR”. see cmplxins.sql

5. Display all location or division or department names eliminating duplicates. The company is looking at defining a new set of specifications on the names it uses so it needs to look at all the current alphanumeric names being used. Eliminate duplicates. See union.sql Use the ORG table.

6. You have received a promotion. Upgrade your salary by 10%. Give yourself a $5,000.00 commission as well. See update6.sql. Use the STAFF table

7. Find the person(s) who has worked with the company longest and give them a 10% salary increase. See update7.sql Use the STAFF table.

8. Whoops!!. Our applicant tables need to be cleaned up. Delete anyone in the applicant tables who is now a permanent employee. Please delete them from the appropriate tables. See cmplxdel.sql