SQL: CREATE TABLES

Naming Rules

Table names and column names:

- Must begin with a letter
- Must 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 an ORACLE server reserved word

Guidelines

Use descriptive names for tables and other database objects.

SYNTAX

```
CREATE TABLE table
     ( column  datatype  [DEFAULT expre]  [, ...]);
```

You need to specify the table name, column name, column data type and column size.

DEFAULT *expr* specifies a default value if a value is omitted in the INSERT statement
This option prevents null values from entering the columns if a row is inserted without a value for the column.
Example: .... Bdate DATE DEFAULT SYSDATE, ...

- Literal values, expressions or SQL functions are legal values.
- Another column's name or a pseudocolumn (NEXTVAL or CURRVAL) are illegal values.
- The default data type must match the column data type.

An automatic commit takes place when this statement is executed.

Example

```
CREATE TABLE Department (
    dname    VARCHAR2(15)
       CONSTRAINT Department_dname_NN  NOT NULL,
     dnumber  INT
       CONSTRAINT Department_dnumber_PK PRIMARY KEY,
     mgrssn     CHAR(9)
       CONSTRAINT Department_mgrssn_NN NOT NULL
       CONSTRAINT Department_mgrssn _FK
             REFERENCES Employee(ssn),
     mgrstartdate  DATE
  )
```

Data Type	**Description**
VARCHAR2(*size*)	Variable-length character data (max *size* must be specified: min *size* = 1; max *size* = 4000)
CHAR [(*size*)]	Fixed-length character data of length *size* bytes(default and min *size* = 1; max *size* = 2000)
NUMBER[(*p,s*)]	Number having precision *p* (total number of decimal digits; 1 <= p <= 38) & scale *s* (number of digits to the right of decimal pt; -84 <= s <= 127)
DATE	Date and time values to the nearest second between Jan 1, 4712 BC and Dec 31, 9999 AD
LONG	Variable-length character up to 2 gigabytes; A LONG column is not copied when a table is created using a subquery; cannot be included in a GROUP BY or an ORDER BY clause; ONLY ONE LONG column can be used per table; NO CONSTRAINTS can be defined on a LONG column (use a CLOB column instead of LONG)
CLOB	Character data up to 4 GB
RAW (*size*)	Raw binary data of length *size* (max *size* must be specified:; max *size* = 2000)
LONG RAW	Raw binary data of variable length up to 2 GB
BLOB	Binary data up to 4 GB
BFILE	Binary data stored in an external file; up to 4 GB
ROWID	A 64 base number system representing the unique address of a row in its table

Refer to DATES and CONSTRAINTS notes for more details.

If you have put FOREIGN KEY constraint clauses in your CREATE TABLE commands, then the order of issuing CREATE TABLE commands will matter.

In Company DB

TABLE	**FOREIGN KEY CONSTRAINTS**	**REFERRED TABLE**
EMPLOYEE	SUPERSSN DNO	EMPLOYEE (SSN) **DEPARTMENT**(DNUMBER)
DEPARTMENT	MGRSSN	EMPLOYEE(SSN)
PROJECT	DNUM	**DEPARTMENT**(DNUMBER)
DEPENDENT	ESSN	EMPLOYEE(SSN)

WORKS_ON	ESSN
	PNO
		EMPLOYEE(SSN)
		PROJECT(PNUMBER)

DEPT_LOCATIONS	DNUMBER	**DEPARTMENT**(DNUMBER)

The order of executing the CREATE TABLE EMPLOYEE|DEPARTMENT|PROJECT|DEPENDENT|WORKS_ON|DEPT_LOCATIONS is as follows

1. CREATE TABLE EMPLOYEE without the FK for DNO
2. a. CREATE TABLE DEPARTMENT NOTE: 2a and 2b can be interchanged
   b. CREATE TABLE DEPENDENT
3. CREATE TABLE DEPT_LOCATIONS NOTE: 3 & 4 can be interchanged but 3 must be after 2a
4. CREATE TABLE PROJECT NOTE: 4 must be after 2a
5. CREATE TABLE WORKS_ON NOTE: 5 must be after 1 and 4
6. ALTER TABLE EMPLOYEE to add FK for DNO

```
    ALTER TABLE EMPLOYEE
    ADD CONSTRAINT Employee_Dno_FK FOREIGN KEY (Dno)
         REFERENCES  Department (Dnumber) [ENABLE/DISABLE];
```

The DISABLE keyword is optional. If you create a constraint using the DISABLE keyword, the constraint will be created, but the condition will not be enforced

For details of the CREATE TABLE commands for the COMPANY tables, refer to ~jmendoza/CS572F09/COMPANY/create_<tablename>.sql files where <tablename> refers to the tables in the COMPANY db.

CREATING A TABLE by USING a SUBQUERY

- Create a table and insert rows

```
CREATE TABLE table
        [ (column, column ...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns
- Define the columns with column names and default values.
- Subquery is the SELECT statement that defines the set of rows to be inserted into the new table
- The table is created with the specified column names, and the rows retrieved by the SELECT statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If no column specs are given, the column names of the table are the same as the column names in the subquery.
- The integrity rules are not passed onto the new table, only the column data type definitions.
- Be sure to give a column alias when selecting an expression.

EXAMPLE:

```
CREATE TABLE Dept4
AS
     SELECT SSN, lname,  salary * 12 ANNSAL
     FROM jmendoza.employee
     WHERE dno =4;
```

The expression salary * 12 is given the alias ANNSAL. Without the alias, an error is generated: "*ORA-00998: must name this expression with a column alias.*"

SQL: CONSTRAINTS

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.

Data Integrity Constraints

Constraint	Notation	Description
NOT NULL	NN	Column cannot contain a null value
UNIQUE	UK	Column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	PK	Uniquely identifies each row of the table
FOREIGN KEY	FK	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	CK	Specifies a condition that must be true

- If you do not name a constraint, ORACLE server generates a name using SYS_C*n* format
- STRONGLY recommended to use constraint name so it will be easy to reference them and use meaningful names. SUGGESTED naming convention for constraint name:

  tablename_attribute_Notation

  where *tablename* is the name of the table that has the constraint
  attribute is the name of the column in the table that has the constraint
  Notation is either NN, UK, PK, FK, CK
- Create a constraint either
  - At the same time as the table is created or
  - After the table has been created
- Define a constraint at the column or table level

  Always use a table level constraint for a primary key that is composite (ie. more than one attribute).
- A table can have only one PRIMARY KEYconstraint but can have several UNIQUE constraints.
- A UNIQUE index is automatically created for a PRIMARY KEY column.

EXAMPLE of a COLUMN CONSTRAINT LEVEL

```
CREATE TABLE Department (
   dname     varchar2(15)  constraint Department_dname_NN NOT NULL
      constraint Department_dname_UK UNIQUE,
  dnumber  int          constraint Department_dnumber_PK PRIMARY KEY,
  mgrssn      char(9)  constraint Department_mgrssn_NN NOT NULL
       constraint Department_mgrssn_FK REFERENCES Employee(ssn),
  mgrstartdate  date
)
```

EXAMPLE of a TABLE CONSTRAINT LEVEL

```
CREATE TABLE Works_On
 ( ESSN  char(9)  CONSTRAINT Works_On_ESSN_FK
   REFERENCES Employee(ssn)                                /* column level
                                                              constraint
   [ON DELETE CASCADE |
    ON DELETE SET NULL ]
   PNO   int ,
   HOURS  decimal(3,1)  CONSTRAINT Works_On_Hours_NN NOT NULL,

  CONSTRAINT Works_On_ESSN_PNO_PK PRIMARY KEY(ESSN,PNO),
                                                            /* table level
                                                              constraint
  CONSTRAINT Works_On_PNO_FK FOREIGN KEY (PNO)
     REFERENCES Project(pnumber)                          /* table level
                                                             constraint
     [ON DELETE CASCADE |
      ON DELETE SET NULL ]
);
```

FOREIGN KEY CONSTRAINT KEYWORDS

• FOREIGN KEY	Defines the column in the child table at the table constraint level
• REFERENCES	Identifies the table and column in the parent table
• ON DELETE CASCADE	deletes the dependent rows in the child table when a row in the parent table is deleted
• ON DELETE SET NULL	converts the dependent foreign key values to NULL

The default behavior is RESTRICT rule, which disallows the update/deletion of referenced data.

WITHOUT the ON DELETE CASCADE or ON DELETE SET NULL options, the row in the parent cannot be deleted if it is referenced in the child table.

CHECK CONSTRAINT

- Defines a condition that each row must satisfy
- The following expressions are not allowed
    - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
    - Calls to SYSDATE, UID, USER, and USERENV functions
    - Queries that refer to other values in other rows
- A single column can have multiple CHECK constraints which refer to the column in its definition. There is no limit to the number of CHECK constraints which you can define at the column level or table level.

EXAMPLE:

```
CREATE TABLE Employee (
   ...
   Salary  NUMBER(8,2)
        CONSTRAINT Employee_Salary_Positive_CK
                   CHECK (Salary > 0)
        CONSTRAINT Employee_Salary_Range_CK
                   CHECK (Salary  BETWEEN 10000 AND 200000),
   ...
```

ADDING A CONSTRAINT

```
ALTER TABLE table
ADD [CONSTRAINT constraintname]   type  ( column ) [DISABLE];
```

where

type is the constraint type (PRIMARY KEY, NOT NULL, FOREIGN KEY, UNIQUE, CHECK)

NOTES:

- You can add/drop/enable/disable a constraint, but you cannot modify its structure.
- You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE command.
- You can define a NOT NULL column only of the table is EMPTY or if the column has a value for every row.

DROPPING A CONSTRAINT

ALTER TABLE *table*
DROP PRIMARY KEY | UNIQUE (*column)* **|**
CONSTRAINT *constraintname* **[CASCADE];**

- You can identify constraint name from the USER_CONSTRAINTS and USER_CONS_COLUMNS data dictionary views.
- The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.
- When you drop an integrity constraint, that constraint is no longer enforced by the ORACLE server and is no longer available in the data dictionary.

DISABLING A CONSTRAINT

ALTER TABLE *table*
DISABLE CONSTRAINT *constraintname* **[CASCADE];**

- The DISABLE clause deactivates an integrity constraint without dropping it or re-creating it.
- You can use the DISABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.
- The CASCADE clause disables dependent integrity constraints.
- Disabling a unique or primary key constraint removes the unique index.

ENABLING A CONSTRAINT

ALTER TABLE *table*
ENABLE CONSTRAINT *constraintname* ;

- The ENABLE clause activates an integrity constraint currently disabled without dropping it or re-creating it.
- Enabling a unique or primary key constraint will automatically create a unique index or primary key index.
- You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.
- Enabling a PK constraint that was disabled with the CASCADE option does not enable any FKs that are dependent upon the PK.

VIEWING CONSTRAINTS

Query the USER_CONSTRAINTS table to view all constraint definitions and names.

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'EMPLOYEE';
```

VIEWING COLUMNS ASSOCIATED WITH CONSTRAINTS

Query the USER_CONS_COLUMNS table to view the columns associated with the constraint names.

This view is especially useful for constraints that use system-assigned names.

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'EMPLOYEE';
```

SQL: ALTER TABLES

Use the ALTER TABLE statement to

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Add a New Column

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

The new column added becomes the last column in the table.
If a table already contains rows when a column is added, then the new column is initially NULL for all the rows.

Modify an Existing Column

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

- You can change a column's data type, size and default value.
  - Increase the width or precision of a numeric column
  - Increase the width of numeric or character columns
  - Decrease the width of a column only if the column contains only null values or if the table has no rows
  - Change the data type only if the column contains null values
  - Can convert a CHAR col to the VARCHAR2 data type or a VARCHAR2 col to the CHAR data type only if the column contains null values or if you do not change the size.
- A change to the default value affects only subsequent insertions to the table.

Drop an Existing Column

```
ALTER TABLE table
DROP          ( column);
```

- Drop columns you no longer need from the table.
- The column may or many not contain data
- Only one column can be dropped at a time.
- The table must have at least one column remaining in it after it is altered.
- Once a column is dropped, it cannot be recovered.

SET UNUSED Option

ALTER TABLE *table*
**SET UNUSED (***column*);

Or

ALTER TABLE *table*
SET UNUSED COLUMN (*column*);

ALTER TABLE *table*
DROP UNUSED COLUMNS;

- SET UNUSED marks one or more columns as unused.
- Specifying this clause does not actually remove the target columns from each row in the table (it does not restore the disk space used by these columns). Therefore the response time is faster than if you executed the DROP clause.
- After a column has been marked as unused, you have no access to that column.
- Names and types of columns marked unused will not displayed in DESCRIBE
- You can add to the table a new column with the same name as an unused column.
- SET UNUSED info is stored in the USER_UNUSED_COL_TABS dictionary view.
- The DROP UNUSED COLUMNS removes from the table all cols currently marked as unused and reclaims the extra disk space from unused cols in the table.

DROPPING a TABLE

DROP TABLE *table;*

- All data and structure in the table is deleted.
- All indexes associated with the table are dropped.
- Cannot roll back the DROP TABLE statement (i.e cannot undo!)
- Any views and synonyms remain but are invalid.
- Only the creator of the table can remove a table.

CHANGING the NAME of an OBJECT

RENAME oldname TO newname;

- Change the name of a table, view, sequence, or synonym
- Must be the owner of the object being renamed.

TRUNCATING a TABLE

TRUNCATE TABLE *table;*

- Removes all rows from a table
- Releases the storage space used by that table
- Cannot roll back row removal
- Must be the owner of the table to be able to TRUNCATE
- Removing rows with TRUNCATE is faster than removing them with DELETE
    - TRUNCATE is DDL statement and generates no rollback info
    - TRUNCATE does not fire the delete triggers of the table
    - If the table is the parent of a referential integrity constraint, you cannot truncate the table. Disable the constraint before issuing TRUNCATE.

ADDING COMMENTS to a TABLE

COMMENT ON TABLE *table* **| COLUMN** *table.column*
IS *'text';*

- A comment can be up to 2000 bytes about a column, table, view.
- Comment is stored in the data dictionary and can be viewed in the COMMENTS column of: ALL_COL_COMMENTS; USER_COL_COMMENTS; ALL_TAB_COMMENTS; USER_TAB_COMMENTS
- To drop a comment from the db set 'text' to empty string ('')
  COMMENT ON TABLE *t1* **IS** '';

SQL: DELETE STATEMENT

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

- Specific rows are deleted if you specify the WHERE clause.
- All rows in the table are deleted if WHERE clause is omitted.

DELETING ROWS BASED ON ANOTHER TABLE

Use subqueries in the DELETE statement to remove rows from a table based on values from another table.

EXAMPLE:

```
DELETE FROM employee
WHERE dno = (SELECT dnumber
             FROM department
             WHERE dname LIKE '%Admin%');
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table. That is if the parent record that you attempt to delete has child records, then you receive the "child record found violation ORA-02292'.

SQL: UPDATE TABLE

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

- Modify existing rows
- Update more than one row at a time
- In general, use the primary key to identify a single row.
  Using other columns can unexpectedly cause several rows to be updated.
- Specific row/rows are modified if you specify the WHERE clause.
- If WHERE clause is omitted, all rows in the table are modified.

UPDATING TWO COLUMNS with a SUBQUERY

```
 UPDATE table
  SET    column =  (SELECT  column
                    FROM  table
                    WHERE  condition)
        [,
         column = (SELECT   column
                    FROM  table
                    WHERE  condition) ]
 [WHERE  condition ];
```

Update employee 114's salary and dno to match that of employee 205.

```
UPDATE  employee
SET    salary = ( SELECT salary
                   FROM    employee
                   WHERE   ssn = '205'),
       dno     = ( SELECT dno
                   FROM    employee
                   WHERE   ssn = '205')
 WHERE  ssn = '114';
```

If you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

UPDATING ROWS Based on Another Table

```
UPDATE  copy_emp
SET    dno  = ( SELECT dno
                FROM    employee
                WHERE   ssn = '205'),
WHERE  job_id  = ( SELECT job_id
                   FROM    employee
                   WHERE   ssn = '100');
```

This changes the department number of all employees with employee 100's job ID to employee 205's current department number.

SQL: INSERT TABLES

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

1. Add new rows to a table.
2. Only one row is inserted at a time.
3. Column list is not required but if not listed, the values must be listed according th the default order of the columns in the table, and a value must be provided for each column.
4. For clarity, use the column list.
5. Enclose character and date values within single quote marks; do not enclose numeric values within single quote marks.
6. Specify the NULL keyword in the VALUES list, specify the empty string (' ') in the VALUES list for character strings and dates.
7. ORACLE server automatically enforces all data types, data ranges, and data integrity constraints.
8. Any column that is not listed explicitly obtains a null value in the new row.
9. Common errors that can occur during user input –
   - Mandatory value missing for a NOT NULL column
   - Duplicate value violates uniqueness constraint
   - Foreign key constraint violated
   - CHECK constraint violated
   - Data type mismatch
   - Value too wide to fit in the column

EXAMPLES:

```
1. INSERT INTO department( dnumber, dname, mgrsssn)   /* explicit
      VALUES (6, 'Finance', '345345345');

2. INSERT INTO  department                            /* implicit
      VALUES (6, 'Finance','345345345', NULL);
```

INSERTING SPECIAL VALUES

- **SYSDATE** records the current date and time
  INSERT INTO employee (SSN, bdate)
  VALUES ('595959598', SYSDATE);
- **USER** records the current username.
  INSERT INTO employee (SSN, username)   /* assuming username is a col
  VALUES ('595959598', USER);

CREATING A SCRIPT - SUBSTITUTION VARIABLES

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable name.
- This will allow you to run the same script file over and over, but supply a different set of values each time you run it.

```
INSERT INTO department
VALUES (‘&dname’, &dnumber,  ‘&mgrssn’,’&mgrstartdate’)

Enter value for dname: Finance
Enter value for dnumber: 99
Enter value for mgrssn: 888665555
Enter value for mgrstartdate: 13-NOV-09
old   2: values ('&dname',&dnumber,'&mgrssn','&mgrstartdate')
new   2: values ('Finance',99,'888665555','13-NOV-09')

1 row created.
```

COPYING ROWS from ANOTHER TABLE

- Write INSERT statement with a subquery.

```
INSERT INTO T1( a1, a2, a3)
   SELECT b1, b2, b3
    FROM   T2
   WHERE  b4 LIKE ‘%*’;
```

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.
- To create a copy of the rows of a table, use SELECT * in the subquery.

```
INSERT INTO copy_emp
   SELECT *
   FROM employee;
```

Handout on SQL*LOADER

The SQL*Loader is a utility program that reads operating system text files and converts the contents into fields in a table. To do this, it must be told what format it should expect the external data to be in. This description is stored in a control file, which usually has the file name extension ".ctl".

BASIC SQLLOAD Process

The usual control file looks like this

Option - Append

```
load data
append
into table <tablename>
fields terminated by ","
(<col1>,<col2>, ..., <coln>)
```

where <coli> must be replaced by the attribute of the table

NOTE: do not use < >!

The append option indicates that the target table may or may not already have rows and that you want to load the data into the table without affecting any current table data.

The data file, typically with extension ".dat" contains the data to be loaded into the table. It should be in the format described in the control file.

Typically,

```
data1,data2,data3, ..., datan
```

NOTE: DO NOT enclose character data types in ''!

To run the loader, type at the delphi operating system prompt, the following command: sqlldr control=<control_file_name>

The system will prompt for username:
Enter your <oracle_userid>/<oracle_password>
DO NOT USE < >!

Note that this command assumes that the control file is in the current directory when the command sqlldr is entered. Otherwise a more complete directory pathname would be needed. Also, SQL*Loader requires write permission in the directory containing the control file, so it can create log files and bad files.
Log files ending with extension ".log" contain a detailed report of the loading process. The bad file with extension ".bad" contains any records that could not be properly read.

You can view these files (*.log and *.bad) by using any text editor.

VARIATIONS on the CONTROL FILE

Option - Insert

```
load data
insert
into table <tablename>
fields terminated by ','
(<col1>,<col2>, ... <coln>)
```

The insert option indicates that the target table is empty.
Use this option if you just created a table definition and the table has no data yet.
If the table is not empty, SQL*LOADER returns an error and cancels the load.

Option - Replace

```
load data
insert
into table <tablename>
fields terminated by ','
(<col1>,<col2>, ... <coln>)
```

The replace option indicates that you want to delete all rows in the table before inserting the data from the load.

All the three options of the control file allows you to use the same control file for different data files.

MORE DETAILS on the SQL*Loader Command Line Options

```
sqlldr [userid=<oracleid>/<oraclepasswd>] control=<fn[.ctl]> data=<fn[.dat]>
           [LOG=<fn>] [BAD=<fn>]
```

NOTE: Anything enclosed in brackets [] is optional

userid= : specifies the name and password of the user to connect to ORACLE when performing a data load. Be sure to specify a user who has the privilege to select and insert records into the target table. Additionally, if the control file specifies the REPLACE option, be sure the user you specify also has the privilege to delete records from the target table.
If this option is not specified, the program prompts you for your userid and you enter youruserid/yourpasswd

control= : specifies the name of the control file to use to perform the data load.
The filename must be a valid operating system filename.
If the .ctl extension is not provided, SQLLOAD will look in the current directory for file with name <fn> and extension .ctl

data= : specifies the name of the data file that contains the data to load into the target table. The filename must be a valid operating system filename. If the .dat extension is not provided, SQLLOAD will look in the current directory for file with name <fn> and extension .dat

log= : specifies the name of the file in which to log information about the data load. If this option is not used, SQL*LOADER automatically creates a log file that has the same name as the control file, with extension .log

bad= : specifies the name of the file in which to store data records that are improperly formatted or records that SQL*Loader cannot insert because of errors during the data load.
If this option is not used, SQL*LOADER automatically creates a bad file that has the same name as the control file, with extension .bad

NOTE: The SQL*LOADER command-line options CONTROL, DATA, LOG and BAD are not case sensitive; however the filenames are in UNIX OS.

AGGREGATING DATA USING GROUP FUNCTIONS

Group functions operate on sets of rows to give one result per group. These sets may be the whole table or the table split into groups.

Group Functions

Function	Description
AVG([DISTINCT | ALL] *n*)	Average value of *n,* ignoring null values
COUNT({ * | DISTINCT | ALL*] expr})*	Number of rows, where *expr* evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT | ALL] *expr)*	Maximum value of *expr*, ignoring null values
MIN([DISTINCT | ALL] *expr)*	Minimum value of *expr*, ignoring null values
STDDEV([DISTINCT | ALL] *x*)	Standard deviation of *n*, ignoring null values
SUM ([DISTINCT | ALL] *n*)	Sum values of *n*, ignoring null values
VARIANCE([DISTINCT | ALL] *x*)	Variance of *n*, ignoring null values

GUIDELINES

- DISTINCT makes the function consider only nondupliate values; ALL makes it consider every value including duplicates. The default is ALL and therefore does not need to be specified.
- The data types for the functions with an *expr* argument may be CHAR, VARCHAR2, NUMBER or DATE.
- All group functions ignore null values.
- The ORACLE server implicitly sorts the result set in ascending order when using a GROUP BY clause. To override this default ordering, DESC can be used in an ORDER BY clause.
- You can use MIN and MAX for any data type.
- COUNT(*) returns the number of rows in a table.
- COUNT(expr) returns the number of rows with non-null values for the expr.
- COUNT(DISTINCT *expr*) returns the number of unique, non-null values in the column identified by *expr*.

EXAMPLES

1.
   ```
   SELECT AVG(SALARY), MAX(SALARY), MIN(SALARY), SUM(SALARY)
   FROM jmendoza.employee;
   ```

2.
   ```
   SELECT MIN(bdate), MAX(bdate)
   FROM jmendoza.employee;
   ```

3.
   ```
   SELECT COUNT (*)
   FROM jmendoza.employee
   WHERE dno = 4;
   ```

4.
   ```
   SELECT COUNT(dno)
   FROM jmendoza.employee;
   ```

5.
   ```
   SELECT COUNT(distinct dno)
   FROM jmendoza.employee;
   ```

GROUP FUNCTIONS and NULL VALUES

Group functions ignore null values in the column. For example if we added a COMMISSION in the EMPLOYEE table and only four of the employees have commissions then

```
SELECT AVG(COMMISSION)
FROM jmendoza.EMPLOYEE;
```

This will calculate the total commission paid to all employees divided by the number of employees receiving a commission (four).

USING NVL FUNCTION with GROUP Functions

NVL forces group functions to include null values.

```
SELECT AVG(NVL(COMMISSION, 0))
FROM jmendoza.employee;
```

The average is calculated based on *all* rows in the table, regardless of whether null values are stored in COMMISSION column. The average is calculated as the total commission paid to all employees divided by the total number of employees in the company (let's say 20).

NVL FUNCTION

Converts a null to an actual value.

- Data types that can be used are date, character, and number.
- Data types must match:
    - NVL (COMMISSION, 0)
    - NVL (dbate '01-JAN-97')
    - NVL(superssn, 'No Boss ')   /superssn is char(9)

SYNTAX

NVL (*expr1, expr2*)

expr1 - source value or expression that may contain a null
expr2 - target value for converting the null

CREATING GROUPS OF DATA

At times, it is needed to divide the table of information into smaller groups. This can be done by GROUP BY clause.

SYNTAX

```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

This divides rows in a table into smaller groups by using the GROUP BY clause.

GUIDELINES:

- If you include a group function in a SELECT clause, you cannot select individual results as well, unless the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the columns in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using the ORDER BY clause.

EXAMPLES:

1.
   ```
   SELECT dno, AVG(salary)
   FROM jmendoza.employee
   GROUP BY dno;
   ```

   *This calculates the average salary for **each** department.*

2.
   ```
   SELECT AVG(salary)
   FROM jmendoza.employee
   GROUP BY dno;
   ```

   This can be done but results are not meaningful.

3.
   ```
   SELECT dno, AVG(salary)
   FROM jmendoza.employee
   GROUP BY dno
   ORDER BY AVG(salary)
   ```

4.
   ```
   SELECT dno, AVG(salary) “Average Salary”
   FROM jmendoza.employee
   GROUP BY dno
   ORDER BY “Average Salary”
   ```

GROUPS WITHIN GROUPS

Let’s assume that employee table has job_id column

```
SELECT dno, job_id, sum(salary)
FROM jmendoza.employee
GROUP BY dno, job_id;
```

The select statement computes the sum of the salaries for job-ids within each dno group.

RESTRICTING GROUP RESULTS – HAVING CLAUSE

Use the HAVING clause to restrict groups:

1. Rows are grouped.
2. Group function is applied.
3. Groups matching the HAVING clause are displayed.

SYNTAX:

```
SELECT        column, group_function
FROM          table
[WHERE        condition]
[GROUP BY     group_by expression]
[HAVING       group_condition]
[ORDER BY     column];
```

EXAMPLE:

Query: Find the maximum salary of each department, but show only the departments that have a maximum salary of more than $10,000.

```
SELECT dno, MAX(salary)
FROM jmendoza.employee
GROUP BY dno
HAVING MAX(salary) > 10000;
```

NESTING GROUP FUNCTIONS

```
SELECT  MAX(AVG(SALARY)
FROM jmendoza.employee
GROUP BY dno;
```

NOTE: Group functions can be nested to a depth of two.

Character Functions

1. Case Manipulation Functions (LOWER, UPPER, INITCAP)
2. Character Manipulation Functions (CONCAT, SUBSTR, LENGTH, INSTR, LPAD | RPAD, TRIM, REPLACE

Function	Purpose	Example
LOWER (*column/expression*)	Converts alpha character values to lowercase	LOWER(‘SQL Course’) = sql course
UPPER (*column/expression*)	Converts alpha character values to uppercase	UPPER(‘SQL Course’) = SQL COURSE
INITCAP(*column/expression*)	Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase	UPPER(‘SQL Course’) = Sql Course
CONCAT(*col1/expr1, col2/expr2*)	Concatenates the first character value to the second character value ; equivalent to concatenation operator (||)	CONCAT(‘Hello’,’World’) = HelloWorld
SUBSTR(*col/expr, m, [n])*	Returns specified characters from character value starting at character position *m*, *n* characters long (If m isnegative, count starts from the end of the character value. If n is omitted, all characters to the end of the string are returned.	SUBSTR(‘HelloWorld’,1,5) = Hello
LENGTH(*column/expression*)	Returns the number of characters in the expression	LENGTH(‘HelloWorld’) = 10
INSTR(*col/expr, ‘string’,[,m], [n])*	Returns the numeric position of a named string, Optionally you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the search and report the first occurrence.	INSTR(‘HelloWorld’,’W’) =6
LPAD (*col/expr, n, ‘string’)* RPAD (*col/expr, n, ‘string’)*	Pads the character value right-justified to a total width of n character positions Pads the character value left-justified to a total width of n character positions	LPAD(salary, 10, ‘*’) = *****24000 RLPAD(salary, 10, ‘*’) = 24000*****
TRIM(*leading/trailing/both, trim_character FROM trim_source*	Trim heading or trailing characters (or both) from a character string. If *trim_character or trim_source* is a character literal, you must enclose it in single quotes. Available from Oracle 8i +	TRIM(‘H’ from ‘HelloWorld’) =elloWorld
REPLACE (*text, search_string, replacement_string)*	Searches a text expression for a character string and, if found, replaces it with a specified replacement string	REPLACE(‘HelloWorld’,’ll’,’rr’) = HerroWorld

Number Functions

Function	Purpose	Example
ROUND (*column/expression, n*)	Rounds the column, expression, or value to *n* decimal places, or if *n* is omitted, no decimal places. (If *n* is negative, numbers to the left of the decimal point are rounded.	ROUND (45.926,2) = 45.93
TRUNC(*column/expression, n*)	Truncates the column, expression, or value to *n* decimal places, or if *n* is omitted, then *n* defaults to zero.	ROUND (45.926,2) = 45.92
MOD(*m,n*)	Returns the remainder of *m* divided by *n*	MOD(1600,300) = 100

SELECT ROUND(45.923,2). ROUND(45.923,0). ROUND(45.923, -1),
ROUND(45.923, -2)
FROM **DUAL**;

DUAL is a dummy table to use to view results from functions and calculations.

The result of the above ROUND SQL statement is 45.92 46 50 0

SELECT TRUNC(45.923,2). TRUNC(45.923,0). TRUNC(45.923, -1),
TRUNC(45.923, -2)
FROM **DUAL**;

The result of the above TRUNC SQL statement is 45.92 45 40 0

ORACLE JOIN

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

SYNTAX:

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

GUIDELINES:

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance db 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, you need a minimum of *n-1* join conditions. 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.
- Distinguish columns that have identical names but reside in different tables by using column aliases.
- Table aliases can be up to 30 characters in length, but shorter is better.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- The table alias is valid only for the **current** SELECT statement.
- Table aliases should be meaningful.

NOTE:

1. Equijoin are joins where the join condition is an =.
2. Equijoins are also called simple joins or inner joins.
3. This type of join usually involves primary and foreign key complements.

SELF-JOIN

This join involves one table joined to itself.

Example to find the name of an employee's supervisor, you need to join the table EMPLOYEE to itself.

```
SELECT E.lname EMPLOYEE , S.lname   SUPERVISOR
FROM  jmendoza.EMPLOYEE E, jmendoza.EMPLOYEE  S
WHERE E.superssn = S.ssn;
```

SQLPLUS Format Commands -- Readable Reports

Command	Description
COL[UMN] [column option]	Controls column formats
TTI[TLE] [text\OFF\ON]	Specifies a header to appear at the top of each page of the report
BTI[TLE][text|OFF\ON]	Specifies a footer to appear at the bottom of each page of the report
BRE[AK] [ON report_element]	Suppresses duplicate values and divides rows of data into sections by using line breaks

COLUMN Command Options

Syntax: COL[UMN] [{column|alias} {option}]

Option	Description
CLE[AR]	**Clears any column formats**
HEA[DING] text	**Sets the column heading(a vertical line(|) forces a line feed in the heading if you do not use justification.)**
FOR[MAT format	**Changes the display of the column data**
NOPRI[NT]	**Hides the column**
NUL[L] text	**Specifies text to be displayed for null values**
PRI[NT]	**Shows the column**

Create Column Headings

```
COLUMN last_name HEADING ‘Employee|Name’
COLUMN salary  JUSTIFY LEFT FORMAT $99,990.00
COLUMN manager FORMAT 999999999 NULL ‘No Manager’
```

Command	Description
COL[UMN] column	**Displays current settings for the specified column**
COL[UMN]	**Displays the current settings for all columns**
COL{UMN] column CLE[AR]	**Clears the settings for the specified column**
CLE[AR] COL[UMN]	**Clears the settings for all columns**

COLUMN FORMAT MODELS

Element	Description	Example	Result
9	**Single digit-suppression digit**	**999999**	**1234**
0	**Enforces leading zero**	**099999**	**001234**
$	**Floating dollar sign**	**$9999**	**$1234**
L	**Local currency**	**L9999**	**L1234**
.	**Position of decimal point**	**9999.99**	**1234.00**
,	**Thousand \operator**	**9,999**	**1,234**

Example

Create a script file to create a report that displays the job ID, last name, and salary for every employee whose salary is less than $15,000. Add a centered, two-line header that reads “Employee Report” and a centered footer that reads “Confidential”. Rename the job title column to read “Job Category” split over two lines. Rename the employee name column to read “Employee”. Rename the salary column to read “Salary” and format it as $2,500.00.

```
vi employee_report.sql
SET FEEDBACK OFF
TTITLE ‘Employee|Report’
BTITLE ‘Confidential’
BREAK ON job_id
COLUMN job_id HEADING ‘Job|Category’
COLUMN salary HEADING ‘Salary’ FORMAT $99,999.99
/*   Insert SELECT Statement */
SELECT job_id, last_name, salary
FROM employees
WHERE salary < 15000
ORDER BY job_id, last_name
/
/*  Clear all formatting commands
SET FEEDBACK ON
COLUMN job_id  CLEAR
COLUMN last_name CLEAR
COLUMN salary CLEAR
CLEAR BREAK
```

USING a SUBQUERY in an INSERT Statement

- Can use a subquery in place of the table name in the INTO clause of the INSERT statement
- The select list of this subquery must have the same number of columns as the column list in the VALUES clause.
- Any rules on the columns of the base table must be followed in order for the INSERT statement to work successfully.

  For example, you cannot put in a duplicate employee ID, nor leave out a value for a mandatory not null column.

```
INSERT INTO
       ( SELECT ssn, lname, bdate, , salary,dno
         FROM    employee
         WHERE   dno= 5)
VALUES  (‘999997777’, ‘Taylor’, ’07-JUN-89’, 25000, 5);
```

USING EXPLICIT DEFAULT VALUES

- **DEFAULT with INSERT**

```
INSERT INTO department (dnumber, dname, mgrssn)
VALUES (300, ‘Engineering’, DEFAULT);
```

- **DEFAULT with UPDATE**

```
UPDATE department
SET mgrssn = DEFAULT  WHERE dnumber = 10;
```

SQL: VIEWS

Views logically represents subsets of data from one or more tables. It is 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.

Views are used

- To restrict data access by displaying selective columns from the table
- To make complex queries (e.g. views can be used to query information from multiple tables without the user knowing how to write a join.)
- To provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- To present different views of the same data according to the user's criteria.

A **simple view** is one that

- Derives data from only one table
- Contains no functions or groups of data
- Can perform DML operations through the view

A **complex view** is one that

- Derives data from many tables
- Contains functions or groups of data
- Does not always allow DML operations through the view

Creating a View

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW *viewname*
[(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT *constraint*]]
[WITH READ ONLY [CONSTRAINT *constraint*]];

In the above syntax:

OR REPLACE	re-creates the view if it already exists
FORCE	Creates the view regardless of whether or not the base tables exist
NOFORCE	Creates the view only if the base tables exist. (This is default)
Viewname	Name of the view
Alias	Specifies names for the expressions selected by the view's query. The number of aliases must match the number of expressions selected by the view.
Subquery	Is a complete SELECT statement. You may use aliases for the columns in the SELECT list.
WITH CHECK	Specifies that only rows accessible to the view can be inserted or

OPTION	updated
Constraint	Is the name assigned to the CHECK option constraint
WITH READ ONLY	Ensures that no DML operations can be performed on this view

Examples: Simple View

Create a view of employees working in dno=4. Display only the employee ssn as ID_NUMBER, last name as NAME, 12 * salary as AnnualSalary

SOLUTION 1:

```
CREATE VIEW salvu4
AS SELECT ssn ID_NUMBER, lname NAME, salary * 12 AnnualSalary
    FROM jmendoza.employee
    WHERE dno = 4;
```

SOLUTION 2:

```
CREATE VIEW salvu4( ID_NUMBER, NAME, AnnualSalary)
AS SELECT ssn, lname, salary * 12
    FROM jmendoza.employee
    WHERE dno = 4;
```

Examples: Complex View

Create a complex view of department names, minimum salaries, maximum salaries, and average salaries by department.

```
CREATE VIEW dept_sum_vu  (dept, minsal, maxsal, avgsal)
AS SELECT d.dname, MIN(e.salary), MAX(e.salary), AVG(e.salary)
    FROM jmendoza.employee e, jmendoza.department d
    WHERE  e.dno = d.dnumber
    GROUP BY d.dname;
```

Displaying the Structure of a View

```
desc  salvu4
```

Retrieving Data from a View

```
SELECT *
FROM salvu4;
```

Drop a View

```
DROP VIEW viewname;
```

QUERYING a VIEW

When you access data using a view, the ORACLE server does the following:

1. It retrieves the view definition from the data dictionary USER_VIEWS.
2. It checks access privileges for the view base table.
3. It converts the view query into an equivalent operation on the underlying base table(s). That is, data is retrieved from, or an update is made to, the base tables.

RULES for PERFORMING DML Operations on a VIEW

- Can perform DML operations on simple views
- Cannot remove a row if the view contains the following
  - group functions
  - a GROUP BY clause
  - the DISTINCT keyword
  - the pseudocolumn ROWNUM keyword
- cannot modify data in a view if it contains
  - group functions
  - a GROUP BY clause
  - the DISTINCT keyword
  - the pseudocolumn ROWNUM keyword
  - columns defined by expressions
- cannot add data through a view if the view includes
  - group functions
  - a GROUP BY clause
  - the DISTINCT keyword
  - the pseudocolumn ROWNUM keyword
  - columns defined by expressions
  - NOT NULL columns in the base tables that are not selected by the view

Using the WITH CHECK OPTION Clause

Example:

```
CREATE or REPLACE VIEW empvu4
AS SELECT *
    FROM jmendoza.employee
    WHERE dno = 4
    WITH CHECK OPTION CONSTRAINT empvu4_ck;
```

NOTE: Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

Example:

```
CREATE or REPLACE VIEW empvu4
AS SELECT *
    FROM jmendoza.employee
    WHERE dno=4
    WITH READ ONLY;

DELETE FROM empvu4
WHERE ssn = '123456789'
Is not allowed!
```

Arithmetic with Dates

Operation	**Result**	**Description**
date + number	date	Adds a number of days to a date
date - number	date	Subtracts a number of days from a date
date - date	number of days	Subtracts one date from another
date + number/24	date	Adds a number of hours to a date

Date Functions

Function	**Description**
MONTHS_BETWEEN(date1, date2)	Number of months between date1 and date2 If date1 is later than date2, result is +; else – The non-integer part of the result represents a portion of the month
ADD_MONTHS(date, n)	Adds n number of calendar months to date; n can be negative and must be an integer
NEXT_DAY(date, 'char'	Finds the date of the next specified day of the week ('char') following date. Char may be number representing a day or a character string
LAST_DAY(date)	Finds the date of last day of the month that contains date
ROUND(date[,'fmt'])	Returns date rounded to the unit specified by the format model fmt. If fmt is omitted, date is rounded to the nearest day.
TRUNC(date[,'fmt'])	Returns date with the time portion of the day truncated to the unit specified by format model fmt. If fmt is omitted, date is truncated t the nearest day

Eample: Display the employee number, hire date, number of months employed, six-month review, first Friday after hire date, and last day of the hire month for all employees employed for fewer than 36 months.

```
SELECT employee_id, hire_date,
          MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
          ADD_MONTHS( hire_date, 6) REVIEW,
          NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
FROM  employees
WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```

EXPLICIT DATA TYPE CONVERSION

NUMBER to CHARACTER → TO_CHAR(number, [fmt])
CHARACTER to NUMBER → TO_NUMBER (char, [fmt])
CHARACTER to DATE → TO_DATE(char, [fmt])
DATE to CHARACTER → TO _CHAR(date,[fmt])

Where fmt can be one of the following

Element	Description
YYYY	Full year in numbers
YEAR	Year spelled out
MM	Two-digit value for month
MONTH	Full name of the month padded with blanks to length of 9 characters
MON	Three-letter abbrev of month
DY	Three-letter abbrev of day of the week
DAY	Full name of the day of the week padded with blanks to length of 9 characters
DD	Numeric day of the month
SCC or CC	Century, server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digit of year
Y,YYY	Year with comma in this position
IYYY, IYY, IY,I	Four, three, two or one digit based on the ISO Format
Q	Quarter of year
DDD or DD or D	Day of year, month or week
J	Julian day; number of days since 31 December 4713 B.C.

TO_CHAR FUNCTION WITH NUMBERS

9	Represents a number
0	Forces a zero to be displayed$
$	Places a floating point dollar sign
L	Uses a floating local currency symbol
.	Prints a decimal point
,	Prints a thousand indicator

EXAMPLE:

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name = 'Ernst';
```

NVL Function – converts a null to an actual value (date, character or number)

NVL(expr1, expr2) where expr1 is source value or expression that may contain a null; expr2 is the target value for converting the null

EXAMPLE

```
NVL(number_column, 9)
NVL(date_column, '01-JAN-95');
NVL)character_column,'Unavailable')
```

EXAMPLE

```
SELECT last_name, salary, NVL(commission_pct,0),
(salary * 12) + (salary * 12 * NVL(commission_pct,o)) ANNUAL_SALARY
FROM employees;
```

```
NVL2(expr1, expr2, expr3) → if expr1 is not null, then NVL2 returns expr2
                            if expr1 is null, then NVL2 returns expr3
```

CASE Expression and DECODE Function

```
CASE expr WHEN comparison_expr1 THEN return _expr1
          [WHEN comparison_expr2 THEN return_expr2
           WHEN comparison_expr3 THEN return_exprn
           ELSE else_expr]
END
```

NOTE: expr, comparison_expr and return_expr must be of the same data type which can be CHAR, VARCHAR2, NCHAR, NVARCHAR2

EXAMPLE

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN ‘IT_PROG’ THEN 1.10 * salary
                   WHEN ‘ST_CLERK’ THEN 1.15 * salary
                   WHEN ‘SA_REP’ THEN 1.20 * salary
       ELSE salary
       END   “Revised Salary”
FROM employees;
```

```
DECODE(col|expression, search1, result1 [,search2, result2, ...] [,default]
```

DECODE decodes expression after comparing it to eachc serrch value. If the expression is the same as search, result is returned. If default value is omitted, a null is returned where a search value does not match any of the result values.

```
SELECT last_name, job_id, salary,
       DECODE( job_id , ‘IT_PROG’ , 1.10 * salary,
                        ‘ST_CLERK’, 1.15 * salary,
                        ‘SA_REP’ , 1.20 * salary,
               salary )
       “Revised Salary”
FROM employees;
```

PATTERN MATCHING with LIKE Operator

‘%’ (percent) is a pattern matching symbol that replaces an arbitrary number of zero or more characters.

‘_’ (underscore) is a pattern matching symbol that replaces a single character.

If an underscore or % is needed as a literal character in the string, the character should be preceded by an ***escape character***, which is specified after the string using the keyword ESCAPE.

EXAMPLE: Q12. Retrieve all employees whose address is in Houston, Texas.

```
SELECT Fname, Lname
FROM    jmendoza.employee
WHERE  address LIKE ‘%Houston, TX%’;
```

EXAMPLE : Q12a. Find all employees who were born during the 1950s
NOTE: DATE FORMAT is DD-MON-YY

```
select fname,lname
from jmendoza.employee
where bdate like '__-___-5_';
```

SQL: Use of Prefix and Alias

- **Use of Prefix**

In SQL the same name can be used for two (or more) attributes as long as the attributes are in different relations. In a query that refers to two or more attributes with the same name, we must qualify the attribute name with the relation name by prefixing the relation name to the attribute name and separating the two by a period. The prefixing is done to avoid ambiguity.

EXAMPLE: Let's assume that the attributes of EMPLOYEE relation in the COMPANY DB were called Dnumber and Name instead of Dno and Lname and the attribute Dname of DEPARTMENT relation was also called Name.

Query 1 - Get name and address of all employees in Research Dept

```
SELECT   Fname, EMPLOYEE.name, Address
FROM     EMPLOYEE, DEPARTMENT
WHERE    DEPARTMENT.Name = ’Research’ AND
         DEPARTMENT.Dnumber =EMPLOYEE.Dnumber;
```

- **Use of Alias**

Ambiguity also arises in the case of queries that refer to the same relation twice. For example,

Query – For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT   E.Fname, E.Lname, S.Fname, S.Lname
FROM     EMPLOYEE  AS E,  EMPLOYEE AS S
WHERE    E.Super_ssn  =S.Ssn;
```

Here E and S are two different copies of the EMPLOYEE relation; E represents the employees in the role of supervisees; S represents employees in the role of supervisors. We join the two tables. There is ***only one*** *EMPLOYEE table, the join condition joins the relation with itself by matching the tuples that satisfy the join condition E.Super_ssn =S.Ssn.*

SUMMARY OF SQL QUERIES

```
SELECT <attribute and function list>
FROM <table list>
[WHERE <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING <group condition>]
[ ORDER BY <attribute list>];
```

ORDERING OF QUERY RESULTS

SQL allows the user to order the tuples in the result of a query by the values of one or more attributes, using the **ORDER BY** clause. The default order is ascending order of values. To specify descending order of values, need to use the keyword **DESC** after the attribute name. To specify ascending order explicitly, use the keyword **ASC**.

QO1: Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

```
SELECT Dname, Lname, Fname, Pname
FROM  DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE Dnumber=Dno AND Ssn=Essn AND Pno=Pnumber
ORDER BY Dname, Lname, Fname;
```

To specify descending order on Dname and ascending order on Lname, Fname then

```
SELECT Dname, Lname, Fname, Pname
FROM  DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE Dnumber=Dno AND Ssn=Essn AND Pno=Pnumber
ORDER BY Dname DESC, Lname ASC , Fname ASC;
```

UNION, INTERSECT, MINUS

EXAMPLE:

```
 /*  Make a list of all project numbers for projects that involve an
     employee whose last name is 'Smith', either as a worker or as a
     manager of the department that controls that project.
 */

(SELECT DISTINCT Pnumber
 FROM jmendoza.PROJECT, jmendoza.DEPARTMENT, jmendoza.EMPLOYEE
 WHERE    Dnum=Dnumber     AND
          MgrSsn=Ssn       AND
          Lname='Smith')

UNION

(SELECT DISTINCT Pnumber
 FROM jmendoza.PROJECT, jmendoza.WORKS_ON, jmendoza.EMPLOYEE
 WHERE   Pno=Pnumber     AND
         Essn=Ssn        AND
         Lname='Smith');

/*  The first SELECT retrieves the projects that involve 'Smith'
    as manager of the dept that controls project, and the
    second retrieves the projects that involve a 'Smith' as a
    worker on the project.
*/
```

ANOTHER SOLUTION Q4A using the IN and OR operators.

```
SELECT DISTINCT  pnumber
FROM jmendoza.project
WHERE pnumber  IN  (SELECT pnumber
                     FROM jmendoza.project, jmendoza.department, jmendoza.employee
                     WHERE dnum=dnumber AND
                              Mgrsnn = ssn     AND
                              lname = 'Smith'
                     )
   OR    pnumber  IN  (SELECT pno
                    FROM jmendoza.works_on, jmendoza.employee
                    WHERE essn = ssn              AND
                             lname = 'Smith'
                    );
```

```
CREATE TABLE tn
( attribute    datatype    CONSTRAINT   tn_an_FK REFERENCES parent(pk) [ON
DELETE {CASCADE|SET NULL}]
   .....
  CONSTRAINT  table_constrainttn_an_FK FOREIGN KEY (attribute_in_this_table)
     REFERENCES parent(pk)  [ON DELETE {CASCADE|SET NULL}]
)
```

NOTE:

- The FK is defined in the child table, and the table containing the referenced column is the parent table.
- ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted..
- ON DELETE SET NULL converts foreign key values to null when the parent value is removed.
- The default behavior is the restrict rule, which disallows the update or deletion of referenced data.
- Without the ON DELETE CASCADE or ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table.

SPOOL

Stores query results in a file, or optionally sends the file to a printer.

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]] | OFF | OUT]
```

Enhancements to the SPOOL Command

The SPOOL command tells SQL*PLUS to send all output to the specified flat file. Think how many times you have used the SPOOL command to save your output for later review. Well, 10G improves the usability of the SPOOL command by adding the following syntax:

- APPEND - Appends output data to an existing file. The command will create a new file if the specified file is not found

- CREATE - Creates a new output file and will return an error if the file already exists

- REPLACE - This is the default option. REPLACE will replace an existing file or create it if it is not found
- OFF - closes the output file

- OUT- closes the file and sends it the printer   DO NOT USE in CS572!