

**Modul Number: 0750362**

**Module Name: Database Applications**

**Teacher: Eman Alnaji**

**Part II**

**SQL (DDL, DML, SELECT)**

**SQL: Structured Query Language**

* Language for describing database schema and operations on tables
* DDL, DML, DCL and TCL are considered sublanguages of SQL

**DDL:** Data Definition Language (CREATE, ALTER, DROP)

**DML:** Data Manipulation Language (INSERT, DELETE, UPDATE)

**DCL:** Data Control Language (GRANT, REVOKE)

**TCL:** Transaction Control Language (COMMIT, ROLLBACK)

**Tables**

* SQL entity that corresponds to a relation
* An element of the database schema

**Company Database**

In this course we will use the Company database as an example of database to create and manipulate.

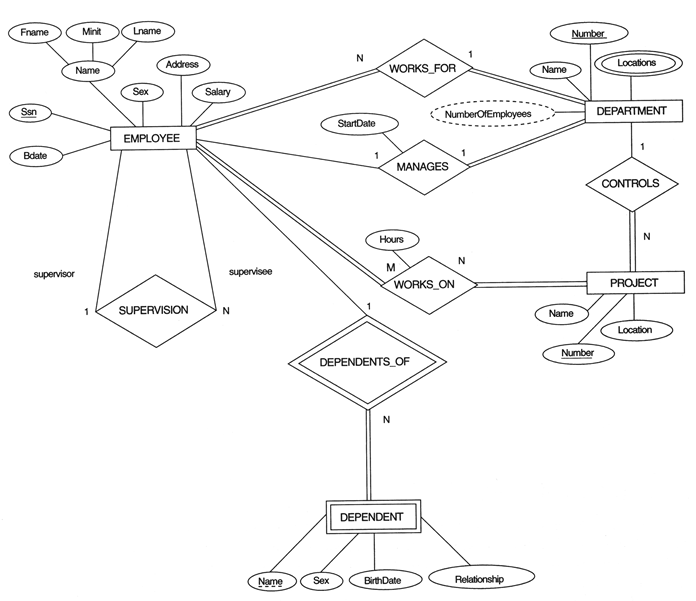
****

Figure1: Company Database ERD.

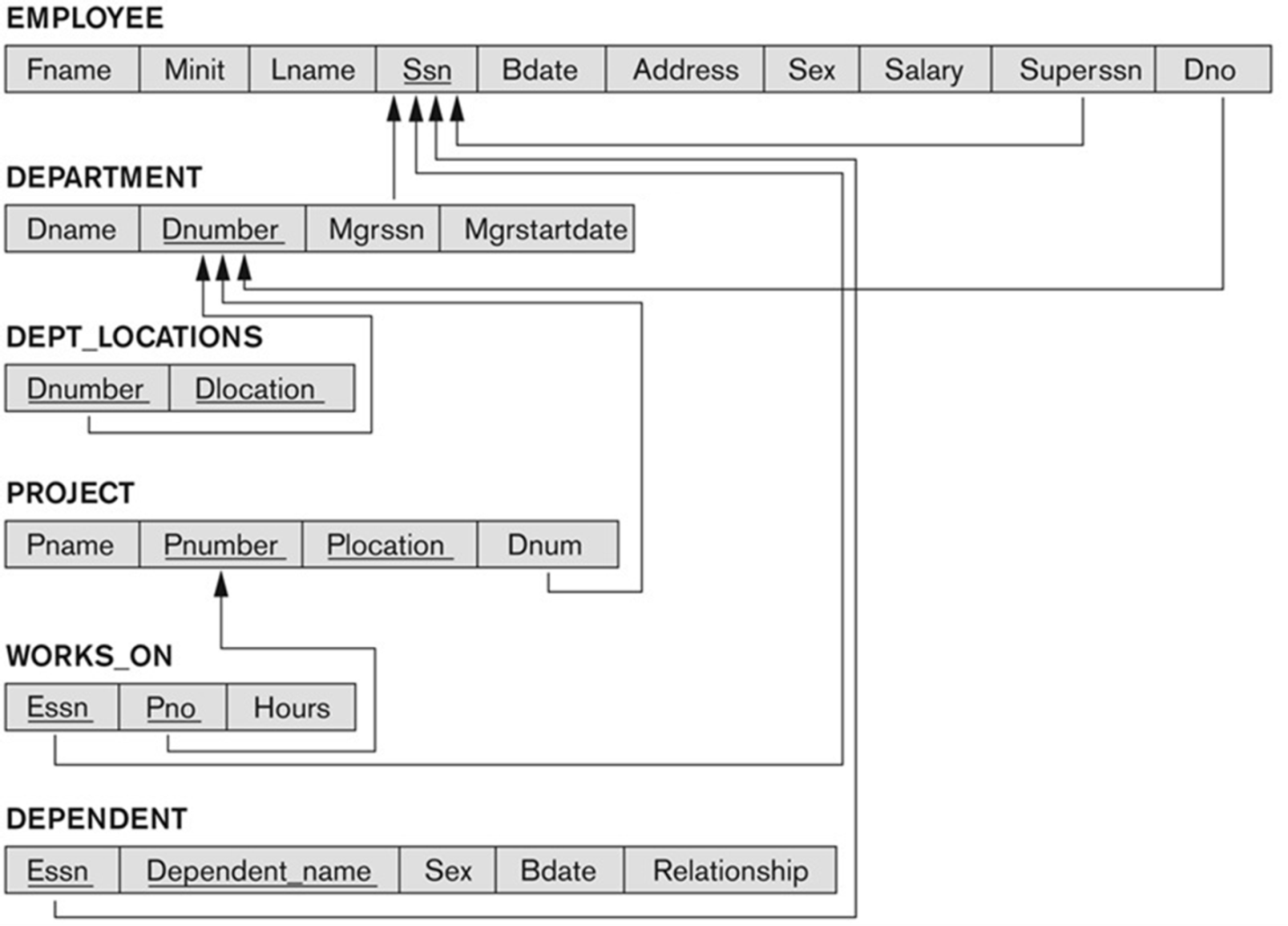
****

Figure2: Company Database Table Schema

**DDL: Data Definition Language**

**Common SQL Data Types (from Oracle):**

String types

CHAR(n) – fixed-length character data, n characters long Maximum length = 2000 bytes

VARCHAR2(n) – variable length character data, maximum 4000 bytes

LONG – variable-length character data, up to 4GB. Maximum 1 per table

Numeric types

NUMBER(p,q) – general purpose numeric data type

Numeric (p, q)- general purpose numeric data type

INTEGER(p) – signed integer, p digits wide

FLOAT(p) – floating point in scientific notation with p binary digits precision

Date/time type

DATE – fixed-length date in dd-mm-yy format

TIME – fixed-length time in hh:mm:ss format

**CREATE TABLE**

Specifies a new base relation by giving it a name, and specifying each of its attributes with their data types.

**A constraint NOT NULL may be specified on an attribute.**

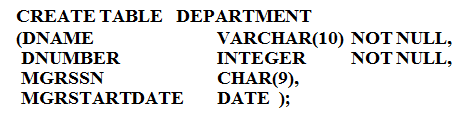


Figure3: The basic CREATE TABLE statement, defining only columns names, data types and NOT NULL constraints.

You can use the CREATE TABLE command for specifying the primary key attributes, candidate keys, and referential integrity constraints (foreign keys).

Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

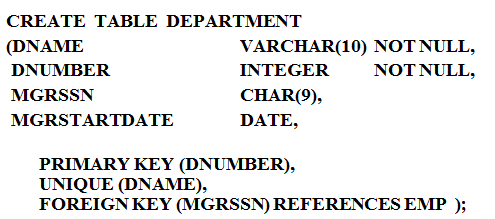


Figure4: CREATE TABLE statement, along with the definition of PRIMARY KEY, UNIQUE key and FOREIGN KEY.

In the previous statement, the definition of a foreign key, will forbid the system from deleting a record from another table that has related records in this table.

But, you can use ON DELETE and ON UPDATE, in the FOREIGN KEY definition, to determine the way of handling the deleting and updating of a record that has related records in this table.

**ON DELETE CASCADE** 🡺 when deleting a related record in another table, delete the related records in the current table.

**ON DELETE SET NULL** 🡺 when deleting a related record in another table, set the foreign key column to NULL (this can be used if the column is defined as NULL).

**ON DELETE SET DEFAULT** 🡺 when deleting a related record in another table, set the foreign key column with the default value determined in the current table. (hint: a default value must be specified in this case).

**ON UPDATE CASCADE** 🡺 when changing the primary key in a related table, then change the foreign key related to it, accordingly.

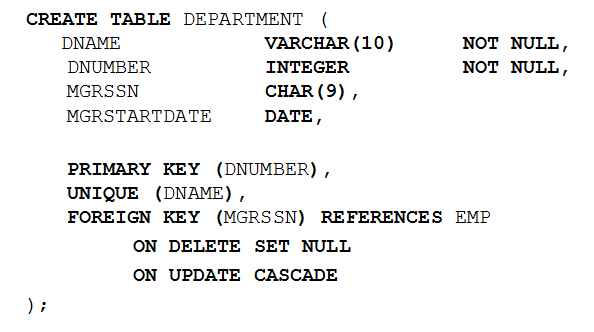


Figure5: Example of ON DELETE and ON UPDATE

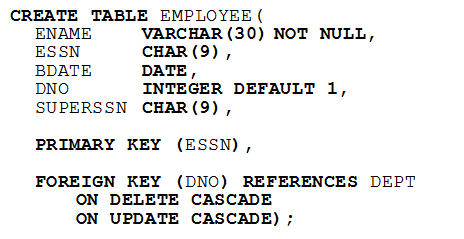


Figure6: Example 2 of ON DELETE and ON UPDATE

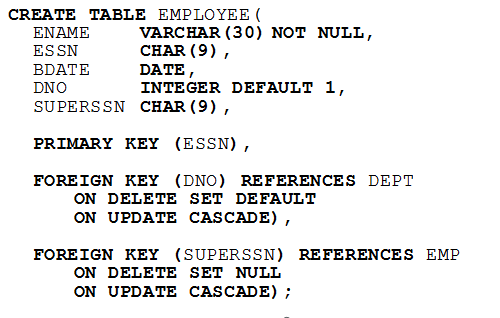


Figure7: Example 3 of ON DELETE and ON UPDATE

**Set DEFAULT values for attributes**

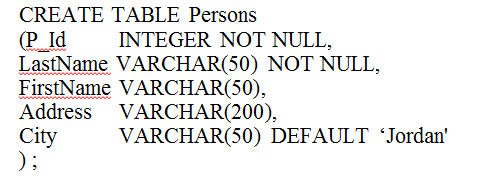


Figure8: Setting default values in table creation

To check the creation of a table, you can view a description of it by using the command DESC.

DESC PERSONS

**ALTER TABLE**

* Used to add an attribute to one of the base relations
* The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute

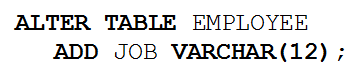


Figure9: Adding a JOB column in table EMPLOYEE

* The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.
* You can change the default value of a certain column, or even set a new default value.



Figure10: Changing the default value of City column

* You can also change the data type or length of an existing column

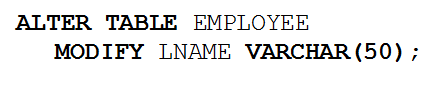


Figure11: Changing the length of column LNAME.

* You can also add a primary key or a foreign key, if you haven’t added them in the create statement.

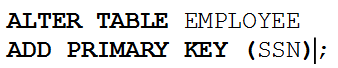


Figure12: Adding a primary key to an existing table

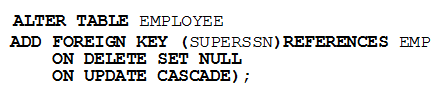


Figure13: Adding a foreign key to table EMPLOYEE

* You can also drop a column that is already existed in a table.

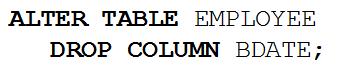


Figure14: Dropping column BDATE from table EMPLOYEE

**DROP TABLE**

* Used to remove a relation (base table) and its definition
* The relation can no longer be used in queries, updates, or any other commands since its description no longer exists



Figure15: Dropping table DEPENDENT

Suppose you need to drop table EMPLOYEE, while it has related records in table DEPENDENT, the system won’t allow dropping it, unless you give order to cancel the constraints between the two tables using the CASCADE keyword.



Figure16: Dropping table EMPLOYEE and dropping all constraints relating this table to other tables.

**DML: Data Manipulation Language**

There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE.**

**INSERT**

In its simplest form, it is used to add one or more tuples to a relation

Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

**Syntax:**

**INSERT INTO *table\_name* (*column,*…, *column*)  
VALUES (*value*, …, *value*);**

* The *column*s are the names of columns you are putting data into, and the *value*s are that data
* String data must be enclosed in single quotes
* Numbers are not quoted
* You can omit the column names if you supply a value for *every* column

**Example:**

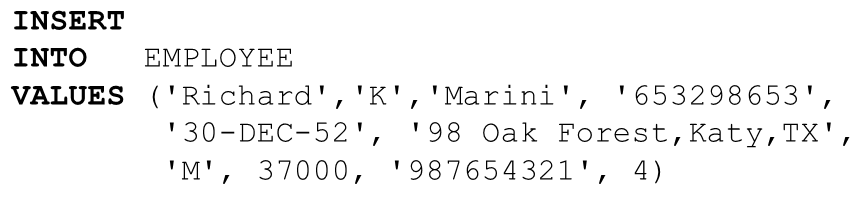


Figure17: Insert Statement

In the previous example, we didn’t specify the names of columns since we are adding values to every column in the table.

The only condition in this case, is to add the values in the order of the columns in the actual table.

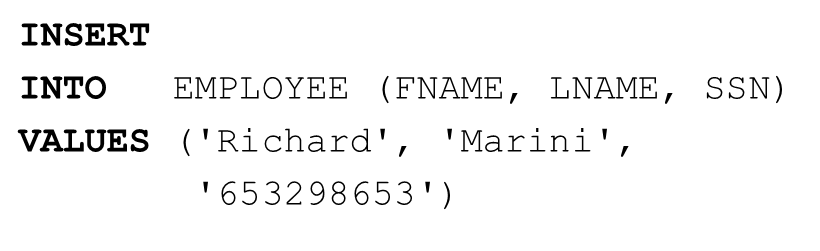


Figure18: Insert Statement

In Figure18, an Insert statement written and some columns are specified. In this case only these columns we have to specify their values. But, we should take into considerations that the other columns in the table can accept NULL values in them, otherwise they should be specified in the Insert Statement.

**Important Note:** Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database. i.e. you cannot insert an Employee in a Department that does not exist, because of the Foreign Key constraint between the two tables.

**DELETE**

* Removes tuples from a relation
* Includes a WHERE-clause to select the tuples to be deleted
* Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
* A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
* The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
* Referential integrity should be enforced

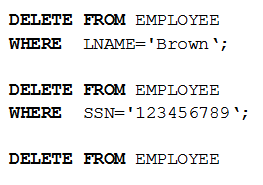


Figure19: Examples on DELETE Statement

**UPDATE**

* Used to modify attribute values of one or more selected tuples
* A WHERE-clause selects the tuples to be modified
* An additional SET-clause specifies the attributes to be modified and their new values
* Each command modifies tuples *in the same relation*
* Referential integrity should be enforced

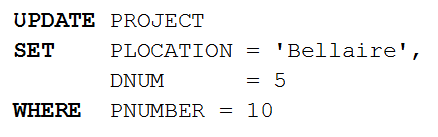


Figure20: Example of UPDATE Statement; updating the location and department number of project of number 10.

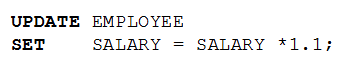


Figure21: Updates the salaries of all Employees by giving them a 10% raise.

**Retrieval Queries in SQL**

* SQL has one basic statement for retrieving information from a database; the SELECT statement
* Basic form of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

**SELECT** <attribute list>

**FROM** <table list>

**WHERE** <condition>

* + <attribute list> is a list of attribute names whose values are to be retrieved by the query
  + <table list> is a list of the relation names required to process the query
  + <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query

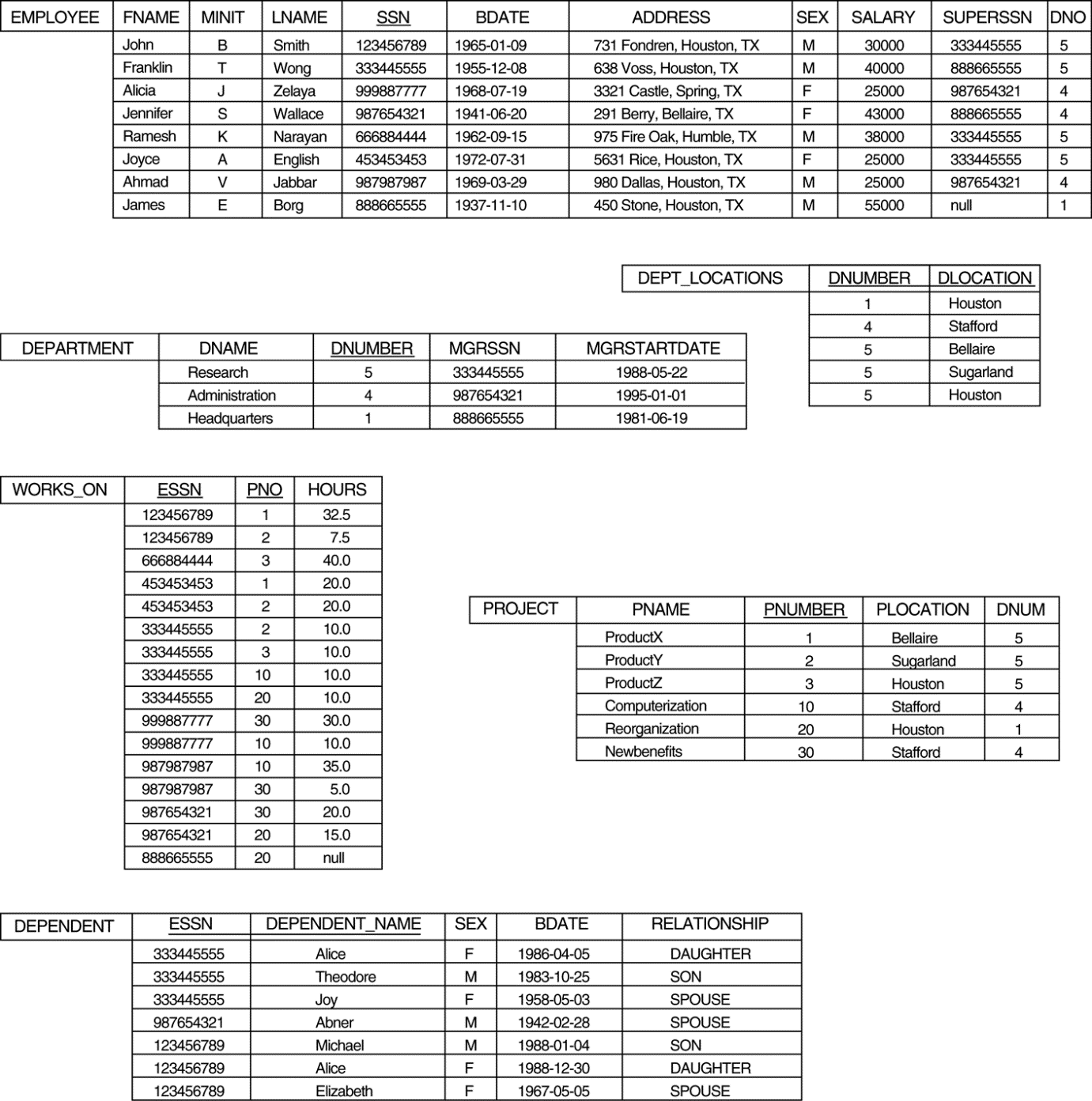


Figure22: The database used in the coming SELECT statements.

In the following pages, SELECT statement will be discussed by examples.

***Query0:*** A simple query on one relation

Retrieve the birthdate and address of the employee whose name is ‘John B. Smith’.

Note here that (birthdate, address and name are all attributes in one relation Employee).

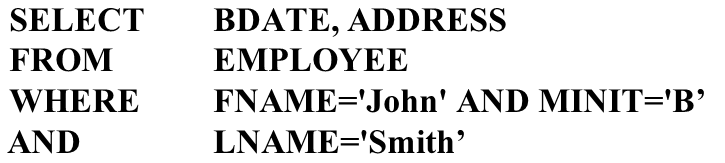


Figure23: Query0

***Query1:*** Retrieve the name and address of all employees who work for the ‘Research’ department.

Note here, that the employee name and address can be accessed from table EMPLOYEE, but the department name does not exist there, we have to use the DEPARTMENT table to find it. So, we need to join both EMPLOYEE and DEPARTMENT tables in the query. (i.e. using the FK).

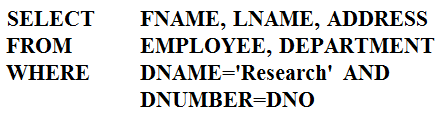


Figure24: Query1

**Note:** (DNAME='Research') is a *selection condition*

(DNUMBER=DNO) is a *join condition*

***Query2:*** For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

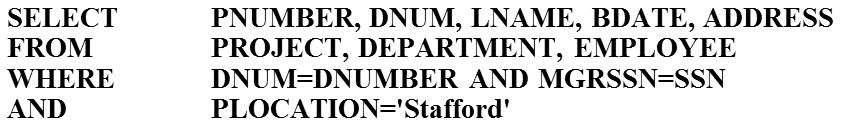


Figure25: Query2

Here we have two Join Conditions:

DNUM=DNUMBER relates a project to its controlling department

MGRSSN=SSN relates the controlling department to the employee who manages that department

**ALIASES**

In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

Example:

DEPT\_LOCATIONS.**DNUMBER**, DEPARTMENT.**DNUMBER**

You can change the name of a table in a query, by giving it an aliases that may be easier to use in a query. Note: That you are not changing the actual name of a table, only giving it another name in the query.

***Query3:*** For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

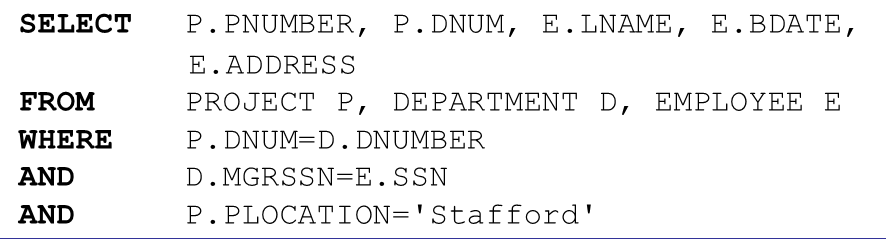


Figure26: Query3

In the previous query, the three tables, PROJECT, DEPARTMENT and EMPLOYEE are given the aliases P, D and E respectively, and they are used to distinguish between the actual tables.

Some queries need to refer to the same relation twice

In this case, *aliases* are given to the relation name

***Query4:*** For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

Note here, that in both cases you will need to retrieve the name of an employee, either it was an ordinary employee or a supervisor. In both cases they reside in the table EMPLOYEE. So, we give two aliases to the EMPLOYEE table, and treat it as two copies of the same table.

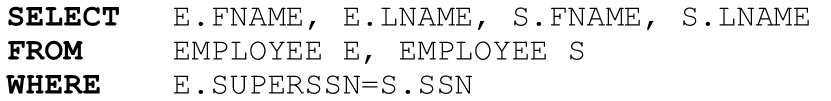


Figure27: Query4

**Unspecified WHERE-Clause**

A *missing WHERE-clause* indicates no condition; hence, *all tuples* of the relations in the FROM-clause are selected.

***Query5:*** Retrieve the SSN values for all employees.

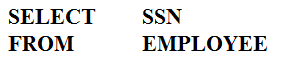


Figure28: Query5

If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected.

***Query6:***



Figure29: Query6

It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result.

**USE OF \***

To retrieve all the attribute values of the selected tuples, a \* is used, which stands for *all the attributes.*

***Query7:*** Retrieve all attributes of all employees who work in department number 5.

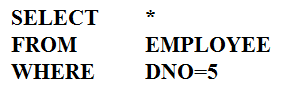


Figure30: Query7

***Query8:*** Retrieve all attributes of all employees along with the attributes of their departments they work for, as long as they work in the ‘Research’ department.

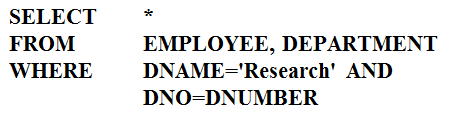


Figure31: Query8

**Use of DISTINCT**

To eliminate duplicate tuples in a query result, the keyword DISTINCT is used.

***Query9:*** Retrieve the salaries of employees without repetition.



Figure32: Query9

**Set Operations**

SQL has directly incorporated some set operations

* UNION
* MINUS
* INTERSECT

The resulting relations of these set operations are sets of tuples

* *duplicate tuples are eliminated* *from the result*

The set operations apply only to *union compatible relations*

* The two relations must have the same attributes and the attributes must appear in the same order

***Query10:*** Make a list of all project names for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

* A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
* Many of the previous queries can be specified in an alternative form using nesting

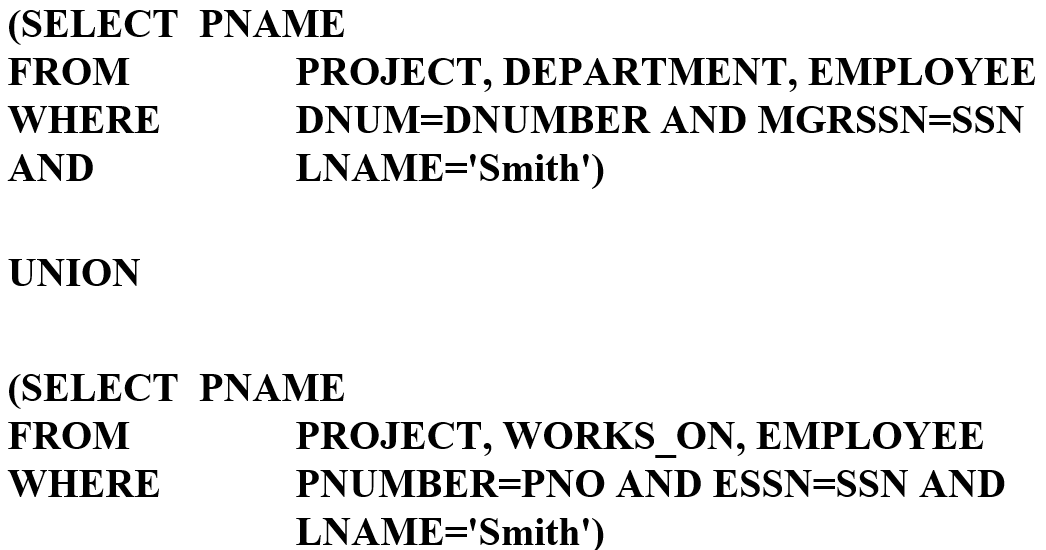


Figure 33: Query10.

**Nesting of Queries**

A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*

Many of the previous queries can be specified in an alternative form using nesting

***Query11:*** Retrieve the name and address of all employees who work for the 'Research' department.

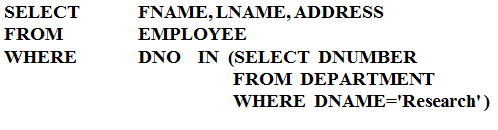


Figure34: Query11

The nested query selects the number of the 'Research' department

The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query

The comparison operator **IN** compares a value v with a set (or multi-set) of values V, and evaluates to **TRUE** if v is one of the elements in V

In general, we can have several levels of nested queries

In the previous example, the nested query is *not correlated* with the outer query.

**Correlated Nested Queries**

If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*

The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

***Query12:*** Retrieve the name of each employee who has a dependent with the same first name as the employee.

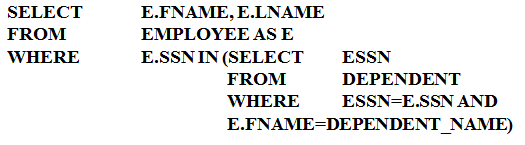


Figure35: Query12

In Query12, the nested query has a different result *for each tuple* in the outer query.

A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query.

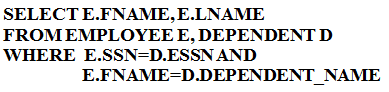


Figure36: Query12 – Rewritten

**The EXISTS Function**

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

We can formulate Query12 in an alternative form that uses EXISTS as in Query13 below.

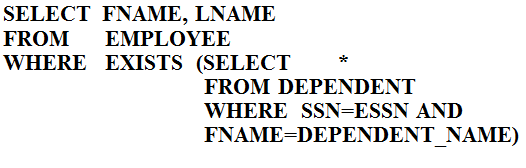


Figure37: Query13.

***Query14:*** Retrieve the names of employees who have no dependents.

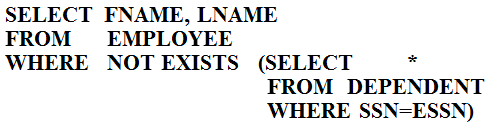


Figure38: Query14

In Query14, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected.

**Explicit SETS**

It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query.

***Query15:*** Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

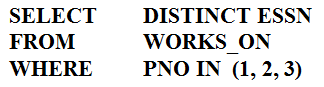


Figure39: Query15.

**NULLs in SQL Queries**

SQL allows queries that check if a value is NULL (missing or undefined or not applicable)

SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .

***Query16:*** Retrieve the names of all employees who do not have supervisors.

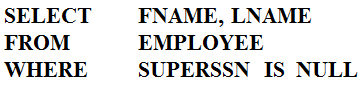


Figure40: Query16

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result.

***Query17:*** Retrieve the names of all employees who have supervisors.

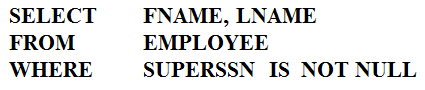


Figure41: Query17

**Aggregate Functions**

Include **COUNT, SUM, MAX, MIN,** and **AVG.**

***Query18:*** Find the maximum salary, the minimum salary, and the average salary among all employees.

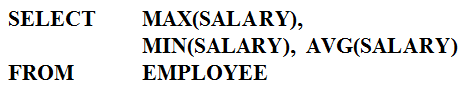
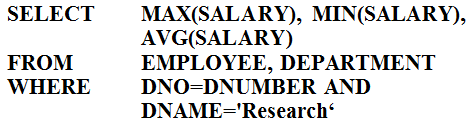


Figure42: Query18.

***Query19:*** Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.



Figrue43: Query19.

***Query20:*** Retrieve the total number of employees in the company.



Figure44: Query20

***Query21:*** Retrieve the total number of employees in the 'Research' department.

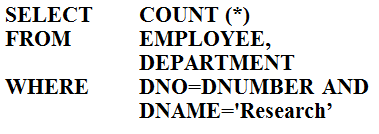


Figure45: Query21

**Grouping**

In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*

Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*

The function is applied to each subgroup independently

SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause.*

***Query22:*** For each department, retrieve the department number, the number of employees in the department, and their average salary.

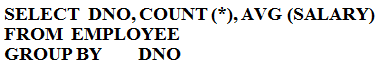


Figure46: Query22

In Query22, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO

The COUNT and AVG functions are applied to each such group of tuples separately

The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples

A join condition can be used in conjunction with grouping.

***Query23:*** For each project, retrieve the project number, project name, and the number of employees who work on that project.

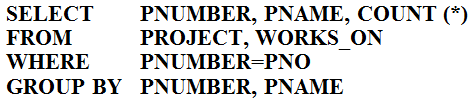


Figure47: Query23

In this case, the grouping and functions are applied *after* the joining of the two relations.

**The HAVING-Clause**

Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples).

***Query24:*** For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.

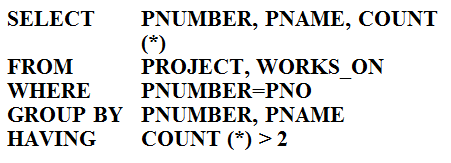


Figure48: Query24

**Substring Comparison**

The **LIKE** comparison operator is used to compare partial strings

Two reserved characters are used: '%' (or '\*' in some implementations) replaces an arbitrary number of characters, and '\_' replaces a single arbitrary character.

***Query25:*** Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.



Figure49: Query25

***Query26:*** Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '\_\_\_\_\_\_\_5\_', with each underscore as a place holder for a single arbitrary character.

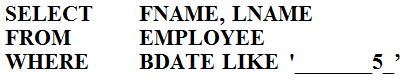


Figure50: Query26

**Arithmetic Operations**

The standard arithmetic operators '+', '-'. '\*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

***Query27:*** Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

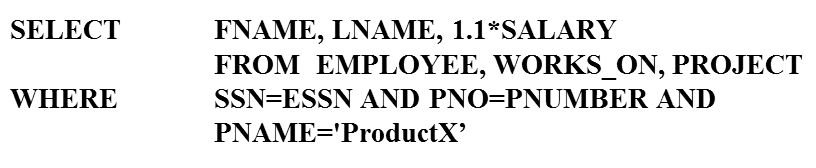


Figure51: Query27

**ORDER BY**

The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)

***Query28:*** Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.



Firgure52: Query28.

The default order is in ascending order of values

We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

**Summary of SQL Queries**

A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

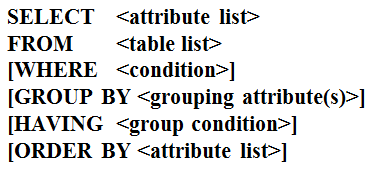


Figure53: Syntax of Select Statement

The **SELECT**-clause lists the attributes or functions to be retrieved

The **FROM**-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries

The **WHERE**-clause specifies the conditions for selection and join of tuples from the relations specified in the **FROM**-clause

**GROUP BY** specifies grouping attributes

**HAVING** specifies a condition for selection of groups

**ORDER BY** specifies an order for displaying the result of a query

A query is evaluated by first applying the **WHERE**-clause, then **GROUP BY** and **HAVING**, and finally the **SELECT**-clause

**SELECT Statement in DML**

Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation.

**Example:** Create a temporary table that has the name, number of employees, and total salaries for each department.

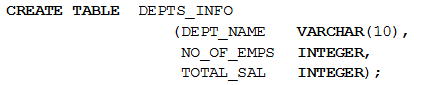


Figure54: Create a new table DEPTS\_INFO

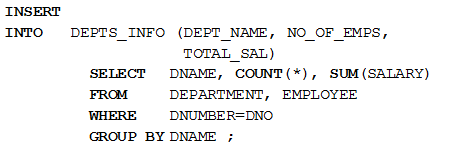


Figure55: Inserting data into DEPTS\_INFO using a SELECT Statement

**Note:** The DEPTS\_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations.

**Example2:** Delete all Employees in ‘Research’ Department.

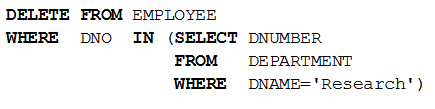


Figure56: Delete from EMPLOYEE table according to a SELECT Statement.