Chapter (8)
SQL The Relational Database Standard

Objectives
• Presenting the main features of SQL
• SQL is the Structured Query Language
• SQL commands

SQL used to be called SEQEL (Structured English QUEry Language). It was designed by the IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL. In 1998 due to a joint effort by ANSI and ISO led to a standard version of SQL called SQL-86 or SQL 1. The revised version of SQL1 is called SQL2 (SQL-92).

SQL is a comprehensive database language. It has statements for data definition, query, and update. Thus, it can be used in both DDL and DML. SQL has rules for embedding SQL statements into programming languages such as C or PASCAL.

Data Definition, Constraints, and Schema Changes in SQL2
SQL uses terms table, row, and column for relation, tuple, and attribute, respectively.

The SQL2 commands for data definition are CREATE, ALTER, and DROP.

Schema and Catalog Concepts in SQL2
Early SQL did not include the concept of a relational database schema. Thus, all tables (relations) were considered part of the same schema.

An SQL schema is identified by a schema name an includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema.

Schema elements include the tables, constraints, views, domains, and other constructs that describe the schema. A schema is created using CREATE SCHEMA statement.
Data Definition, Constraints, and Schema Changes in SQL2
Example of schema creation:
```
CREATE SCHEMA COMPANY AUTHORIZATION JSMITH;
```

The CREATE TABLE Command and SQL2 Data Types and Constraints
The CREATE TABLE command is used to specify a new relation by giving it name and specifying its attributes and constraints.

The attributes are specified first and each attribute is given name, a data type to specify its domain of values, and any attribute constraints such as NOT NULL.

The key, entity integrity, and referential integrity constraints can be specified, within the CREATE TABLE statement, after declaration of variables. They also can be added later using the ALTER command.

The CREATE TABLE Command and SQL2 Data Types and Constraints – cont.

Data Types and Domains in SQL2

Data Types are:

1. Numeric
   - `INTEGER` or `INT`, and `SMALLINT`
   - Real Numbers (`FLOAT`, `REAL`, `DOUBLE PRECISION`)
   - Formatted numbers (`DECIMAL(i,j)` ) or `DEC(i,j)`
     or `NUMERIC(i,j)` – `i` is precision `j` is the number of digits.

2. Character-string
   - `CHAR(n)` or `CHARACTER(n)` – `n` is number of characters
   - `VARCHAR(n)` or `CHAR VARYING(n)` or `CHARACTER VARYING(n)` – `n` is the maximum number of characters.
Data Types and Domains in SQL2

3. Bit-string

BIT(n) for fixed length n or
BIT VARYING(n) – n is maximum number of bits, default is one.

4. Date

YEAR, MONTH, and DAY as YYYY-MM-DD

5. Time

HOUR, MINUTE, and SECOND as HH:MM:SS

Also we may have TIME(i) where i is called time fractional seconds precision, i + 1 additional positions for TIME – one position for an additional separator character, and I positions for specifying decimal fractions of a second.

A CREATE TABLE Example

CREATE procedure for EMPLOYEE table below.

CREATE TABLE EMPLOYEE
(FNAME VARCHAR(15) NOT NULL,
MINIT CHAR,
LNAME VARCHAR(15) NOT NULL,
SSN CHAR(9) NOT NULL,
BDATE DATE,
ADDRESS VARCHAR(30),
SEX CHAR,
SALARY DECIMAL(10,2),
SUPERSSN CHAR(9),
DNO INT NOT NULL,
PRIMARY KEY (SSN),
FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN),
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNUMBER);
In SQL2, it is possible to specify the data type of each attribute directory. Also, a domain can be declared, and the domain named used.

CREATE DOMAIN SSN_TYPE AS CHAR(9)

This way, we can use SSN_TYPE in place of CHAR(9).

**Specifying Constraints and Default Values in SQL2**

Null is allowed in SQL, thus a NOT NULL may be specified if it is not permitted. This must be defined for the primary keys at all times.

It is also possible to define a default value using DEFAULT command.

Perhaps the most important feature is a **referential triggered action**. This option allows user to SET NULL, CASCADE, and SET DEFAULT when a referential integrity constraint is violated by insertion or deletion of a tuple.

The above actions may take place ON DELETE or ON UPDATE.

In general, the action taken by DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE. In such a case, the value of the affecting referencing attributes is changed to NULL for SET NULL, and to the specified default value for SET DEFAULT.

The action for CASCADE on DELETE is to delete all the referencing tuples, whereas the action for CASCADE ON UPDATE is to change the value of the foreign key to the updated (new) primary key value for all referencing tuples.

A database designer decides which action to take.
Data Types and Domains in SQL2 – cont.

A constraint may be given a name, following the keyword CONSTRAINT. The name of all constraints within a particular schema must be unique. The constraint name is used to drop or replace it with another constraint.

The CREATE TABLE command creates the base tables (or base relations). We also can create virtual relations using the CREATE VIEW statement. The view may not necessarily correspond to an actual relation in real world.

The DROP SCHEMA and DROP TABLE Commands

When a schema is no longer needed it maybe deleted using the DROP SCHEMA command. There two drop behavior options: CASCADE and RESTRICT.

Example: To remove the COMPANY database schema and all its tables, domains, and other elements, with CASCADE option:

```
DROP SCHEMA COMPANY CASCADE;
```

The DROP SCHEMA and DROP TABLE Commands – cont.

In the previous example if RESTRICT option was used, the schema would have been dropped only if it has no element in it, otherwise the DROP command will not execute.

IF a base relation is not needed, the relation and its definition can be deleted using DROP TABLE command.

Example:

```
DROP TABLE DEPENDENT CASCADE;
```

If would used RESTRICT instead of CASCADE, the table would get deleted only it is not referenced in any constraints.

Example:

A table can be referenced using the foreign key definition in another relation.
The ALTER TABLE Command
The definition of a table can be changed using the ALTER TABLE command. Possible actions by ALTER are:
- Adding or dropping a column (attribute)
- Changing a column definition, and
- Adding or dropping table constraints.

Example:
ALTER TABLE COMPANY.EMPLOYEE ADD JOB VARCHAR(12);
This will add an attribute to the table EMPLOYEE in the COMPANY database and is used to keep track of jobs of employees.

We must enter a value for the new attribute for each individual employee. This can be define using a default clause. If no default is defined, the default would be NULL upon execution of the command.

The not NULL constraint is not allowed, why?

The DROP SCHEMA and DROP TABLE Commands – cont.
To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.

If CASCADE is used, all constraints and views that reference the column are dropped automatically from the schema, along with the column.

If RESTRICT is used, the command is successful only if no views or constraints reference the column.

Example:
ALTER TABLE COMPANY.EMPLOYEE DROP ADDRESS CASCADE;
This removes the attribute ADDRESS from the EMPLOYEE table.

It is also possible to alter a column definition:

Example:
ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN DROP DEFAULT;
ALTER TABLE COMPANY.DEPARTMENT ALTER MGRSSN SET DEFAULT “333445555”;
The DROP SCHEMA and DROP TABLE Commands – cont.

We can also change the constraints specified on a table by adding or dropping a constraint.

To drop a constraint, that constraint must have been given a name when it was specified.

Example:

```
ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Once this is done we may (can) redefine a replacement constraint by adding a new constraint to the relation. This is specified by the ADD keyword followed by the new constraint, which can be named or unnamed.

Example Queries

General definition of the SELECT query:

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

Example:

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith';
```

Which was the translation of

```
\( \rho \text{BDATE,ADDRESS} (\delta \text{FNAME='John'} \text{AND MINIT='B'} \text{AND LNAME='Smith'} (\text{EMPLOYEE})) \)
```
Example Queries

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

SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO;

Example:
For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birthdate.

SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
PLOCATION='Stafford';

Dealing with Ambiguous Attribute Names and Renaming (Aliasing)

In SQL the same name can be used for two (or more) attributes as long as the attributes are in different relations. If this is the case, and a query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name, to prevent ambiguity. This is done by prefixing the relation name to the attribute name and separating the two by a period.
In the following example, suppose the DNO and LNAME attributes of the EMPLOYEE relation were called DNUMBER and NAME and the DNAME attribute of DEPARTMENT was also called NAME. Then Query (1) will look like this:

SELECT FNAME, EMPLOYEE.NAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.NAME='Research' AND
DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER;
Dealing with Ambiguous Attribute Names and Renaming (Aliasing)

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

Example:

SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;

In this case, we are allowed to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation. An alias can follow the keyword **AS**.

It is also possible to rename the attributes while making the alias:
EMPLOYEE AS E(FN,MI,LN,SSN,BD,ADDR,SEX,SAL,SSSN,DNO)

Unspecified WHERE-Clause and Use of Asterisk (*)

Sometimes we wish to select everything from a table. In such cases * works like a wild card and includes everything in the result. To see the different let’s look at the following 3 queries:

**Q9** (Selects all EMPLOYEE SSNs)

SELECT SSN
FROM EMPLOYEE;

**Q10** (Selects all combinations of EMPLOYEE SSN and DEPARTMENT DNAME)

SELECT SSN, DNAME
FROM EMPLOYEE, DEPARTMENT;

**Q1C:**

SELECT *
FROM EMPLOYEE
WHERE DNO=5;
Unspecified WHERE-Clause and Use of Asterisk (*)

Other examples and uses:

**Q1D** (retrieves all the attributes of an EMPLOYEE and the attributes of the DEPARTMENT for every employee who work in Research department).

**SELECT** *
**FROM** EMPLOYEE, DEPARTMENT
**WHERE** DNAME=’Research’ **AND** DNO=DNUMBER;

**Q10A** (This specifies the CROSS PRODUCT of the EMPLOYEE and DEPARTMENT)

**SELECT** *
**FROM** EMPLOYEE, DEPARTMENT;

---

Tables as Sets in SQL (use of ALL, DISTINCT)

In SQL when you run a query, you may end up getting duplicates. SQL does not automatically removes duplicates. Duplicate eliminations is: 1) expensive, 2) may be desired by the users, and 3) when we apply an aggregate operation we want the duplicate to appear.

**Example:**

<table>
<thead>
<tr>
<th>Q11</th>
<th>Q11A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>30000</td>
</tr>
<tr>
<td>40000</td>
<td>40000</td>
</tr>
</tbody>
</table>

**Q11:**

**SELECT** ALL SALARY
**FROM** EMPLOYEE;

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25000</td>
<td>25000</td>
</tr>
<tr>
<td>43000</td>
<td>43000</td>
</tr>
<tr>
<td>38000</td>
<td>38000</td>
</tr>
<tr>
<td>25000</td>
<td>55000</td>
</tr>
</tbody>
</table>

**Q11A:**

**SELECT** DISTINCT SALARY
**FROM** EMPLOYEE;

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25000</td>
</tr>
<tr>
<td>55000</td>
</tr>
</tbody>
</table>
Tables as Sets in SQL (UNION, EXCEPT, and INTERSECT)
Duplicates are removed from the result of UNION, EXCEPT, and INTERSECT.
Example: List 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 the project.

Q4:
(SELECT DISTINCT PNUMBER
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME=‘Smith’)
UNION
(SELECT DISTINCT PNUMBER
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME=‘Smith’);

What would be the output of this query?
(SELECT DISTINCT PNUMBER
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME=‘Smith’)
EXCEPT
(SELECT DISTINCT PNUMBER
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME=‘Smith’);

What does it mean?
**Substring Comparisons, Arithmetic Operators, and Ordering**

**Like, %, and _**

Example: Retrieve all employees with an address in Houston, TX.

```sql
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE 'Houston,TX';
```

% is used to specify partial strings, i.e. arbitrary number of characters or accepted. On the contrary, (_) replaces a single space.

Example: Find all employees who were born in 1950s.

**Q12A:**

```sql
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE BDATE LIKE '_5_ _ _ _ _ _ _ _';
```

As you may have already noticed, we have taken advantage of the location where the single character of our interest will appear.

---

**Substring Comparisons, Arithmetic Operators, and Ordering**

**Using +, -, *, and /**

Example: List the employees working on 'ProductX' project with a 10\% raise in their salary.

**Q13:**

```sql
SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Example: List all employees with a salary between $30,000 to $40,000

**Q14:**

```sql
SELECT *
FROM EMPLOYEE
WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO = 5;
```
### Substring Comparisons, Arithmetic Operators, and Ordering

**ORDER BY**

The result of a query appears in the order that they are discovered from the tables. Sometimes, we want to order the resulting table by one or more attribute names.

**Q15:** Make a list of employees and the project 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;
```

By default ordering is done in an ascending order (ASC) to order in a descending order we will use (DESC).

Here is a mixed one:

```
ORDER BY DNAME DESC, LNAME ASC, FNAME ASC
```

### Nested Queries – IN Operator

**Q4A:**

```
SELECT DISTINCT PNUMBER
FROM PROJECT
WHERE PNUMBER IN
(SELECT PNUMBER
 FROM PROJECT, DEPARTMENT, EMPLOYEE
 WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
 LNAME='Smith')
OR
PNUMBER IN
(SELECT PNO
 FROM WORKS_ON, EMPLOYEE
 WHERE ESSN=SSN AND LNAME='Smith');
```

The Comparison IN operator compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V.
Nested Queries – IN Operator
The IN operator also can be used to compare a tuple of values in parentheses
with a set or multiset of union-compatible tuples.

Example: Selects the SSN of all employees who work the same (project, hours)
combination on some project that employee ‘John Smith’ (SSN=123456789)
works on.
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE (PNO, HOURS) IN
SELECT PNO, HOURS FROM WORKS_ON WHERE SSN='123456789');

Example: List the name of employees whose salary is greater that the salary of
all the employees in department 5.
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE
WHERE DNO=5);

Other operators in this group are; ANY and SOME. They too can be combined
with >, <, >=, <=, <>.

Dealing with Ambiguous Attribute Names and Renaming (Aliasing)
An example is the following query, which returns the names of employees whose
salary is greater than the salary of all the employees in department 5:

SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE
WHERE DNO=5);

Example: Retrieve the name of each employee who has a dependent with the
same first name and same sex as the employee.
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN
(SELECT ESSN
FROM DEPENDENT
WHERE E.FNAME=DEPENDENT_NAME AND E.SEX=SEX);
Correlated Nested Queries

Whenever a condition in the WHERE-clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.

We can understand a correlated query better by considering that the nested query is evaluated once for each tuple (or combination of tuples) in the outer query.

Example: Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E, DEPENDENT AS D
WHERE E.SSN=D.ESSN AND E.SEX=D.SEX AND E.FNAME=D.DEPENDENT_NAME;
```

Most commercial implementations of SQL do not have this operator. The CONTAINS operator compares two sets of values and returns TRUE if one set contains all values in the other set. Query 3 illustrates the use of the CONTAINS operator.

```
Correlated Nested Queries

Retrieve the name of each employee who works on all the projects controlled by department number 5.

SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ( (SELECT PNO
        FROM WORKS_ON
        WHERE SSN=ESSN)
    CONTAINS
    (SELECT PNUMBER
        FROM PROJECT
        WHERE DNUM=5) );
```
The EXISTS, NOT EXISTS and UNIQUE Functions in SQL

Same example again with EXISTS:

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE EXISTS (SELECT *
               FROM DEPENDENT
               WHERE E.SSN=ESSN AND E.SEX=SEX AND
               E.FNAME=DEPENDENT_NAME);
```

Example: Retrieve the names of employees who have no dependents.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
                   FROM DEPENDENT
                   WHERE SSN=ESSN);
```

List the names of managers who have at least one dependent.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE EXISTS (SELECT *
               FROM DEPENDENT
               WHERE SSN=ESSN)
               AND
               EXISTS (SELECT *
                        FROM DEPARTMENT
                        WHERE SSN=MGRSSN);
```

Query 3 can be written differently:

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT PNUMBER
                   FROM PROJECT
                   WHERE DNUM=5) EXCEPT
                   (SELECT PNO
                    FROM WORKS_ON
                    WHERE SSN=ESSN));
```
The EXISTS, NOT EXISTS and UNIQUE Functions in SQL

Yet another way to write Query 3:
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXISTS
  (SELECT *
   FROM WORKS_ON B
   WHERE (B.PNO IN
     (SELECT PNUMBER
      FROM PROJECT
      WHERE DNUM=5))
   AND
     NOT EXISTS (SELECT *
                   FROM WORKS_ON C
                   WHERE C.ESSN=SSN
                   AND C.PNO=B.PNO));

Explicit Sets and NULLS in SQL

We have seen several queries with a nested query in the WHERE-clause. It is also possible to use an explicit set of values in the WHERE-clause, rather than a nested query. Such a set is enclosed in parentheses in SQL.
Example: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN (1, 2, 3);

SQL allows queries that check whether a value is NULL—missing or undefined or not applicable. However, rather than using = or to compare an attribute to NULL, SQL uses IS or IS NOT. This is because SQL considers each null value as being distinct from every other null value, so equality comparison is not appropriate.

Example: Retrieve the names of all employees who do not have supervisors.
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE SUPERSSN IS NULL;
Renaming Attributes and Joined Tables

It is possible to rename any attribute that appears in the result of a query by adding the qualifier `AS` followed by the desired new name.

```
SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;
```

This could be written as:

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME='Research';
```

Or as:

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN (DEPARTMENT AS DEPT (DNAME, DNO,
MSSN, MSDATE)))
WHERE DNAME='Research';
```

The default type of join in a joined table is an inner join, where a tuple is included in the result only if a matching tuple exists in the other relation.

Renaming Attributes and Joined Tables

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.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME
FROM (EMPLOYEE AS E LEFT OUTEER JOIN EMPLOYEE AS S ON
E.SUPERSSN=S.SSN);
```

The options available for specifying joined tables in SQL2 include INNER JOIN (same as JOIN), LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN. In the latter three, the keyword OUTER may be omitted.

Another way to write Query 3:

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM ((PROJECT JOIN DEPARTMENT ON DNUM= DNUMBER) JOIN
EMPLOYEE ON MGRSSN=SSN)
WHERE PLOCATION='Stafford';
```
Aggregate Functions and Grouping
COUNT, SUM, MAX, MIN, and AVG

The above functions can be used in the SELECT-Clause or in a HAVING-clause.

Example: Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE;
```

If we want to get the preceding function values for employees of a specific department—say the ‘Research’ department—we can write Query 20, where the EMPLOYEE tuples are restricted by the WHERE-clause to those employees who work for the ‘Research’ department.

Example: Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research';
```

Here the asterisk (*) refers to the rows (tuples), so COUNT(*) returns the number of rows in the result of the query. We may also use the COUNT function to count values in a column rather than tuples, as in the next example.

Example: Count the number of distinct salary values in the database.

```
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE;
```

Notice that, if we write COUNT(SALARY) instead of COUNT(DISTINCT SALARY) in Q23, we get the same result as COUNT(*) because duplicate values will not be eliminated, and so the number of values will be the same as the number of tuples.
Aggregate Functions and Grouping
COUNT, SUM, MAX, MIN, and AVG

Example: To retrieve the names of all employees who have two or more dependents (Query 5), we can write:

```sql
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE (SELECT COUNT (*)
      FROM DEPENDENT
      WHERE SSN=ESSN) >= 2;
```

The correlated nested query counts the number of dependents that each employee has; if this is greater than or equal to 2, the employee tuple is selected.

In many cases we want to apply the aggregate functions to subgroups of tuples in a relation, based on some attribute values. For example, we may want to find the average salary of employees in each department or the number of employees who work on each project. In these cases we need to group the tuples that have the same value of some attribute(s), called the grouping attribute(s), and we need to apply the function to each such group independently. SQL has a **GROUP BY-clause** for this purpose.

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

```sql
SELECT DNO, COUNT (*), AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNO;
```

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

```sql
SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME;
```

Sometimes we want to retrieve the values of these functions only for groups that satisfy certain conditions. For example, suppose that we want to modify Query 25 so that only projects with more than two employees appear in the result. SQL provides a **HAVING-clause**, which can appear in conjunction with a GROUP BY-clause, for this purpose. HAVING provides a condition on the group of tuples associated with each value of the grouping attributes; and only the groups that satisfy the condition are retrieved in the result of the query.
Aggregate Functions and Grouping

HAVING

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

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2;
```

Example: For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
SELECT DNUMBER, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND SALARY>40000 AND
DNO IN (SELECT DNUMBER
FROM EMPLOYEE
GROUP BY DNO
HAVING COUNT(*) > 5)
GROUP BY DNUMBER;
```

Notice that, while selection conditions in the WHERE-clause limit the tuples to which functions are applied, the HAVING-clause serves to choose whole groups.

Other SQL Queries – INSERT, DELETE, and UPDATE

```
INSERT INTO EMPLOYEE
VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98 Oak Forest,Katy,TX', 'M', 37000, '987654321', 4);

INSERT INTO EMPLOYEE (FNAME, LNAME, DNO, SSN)
VALUES ('Richard', 'Marini', 4, '653298653');

DELETE FROM EMPLOYEE delete zero tuple
WHERE LNAME='Brown';

DELETE FROM EMPLOYEE delete one tuple
WHERE SSN='123456789';

DELETE FROM EMPLOYEE delete four tuples
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research');

DELETE FROM EMPLOYEE; delete all tuples
```
Other SQL Queries – UPDATE

Example: Change the location and controlling department number of project number 10 to ‘Bellaire’ and 5, respectively, we use U5:

UPDATE PROJECT
SET PLOCATION = ‘Bellaire’, DNUM = 5
WHERE PNUMBER = 10;

UPDATE EMPLOYEE
SET SALARY = SALARY * 1.1
WHERE DNO IN
  (SELECT DNUMBER
   FROM DEPARTMENT
   WHERE DNAME = ‘Research’);