

Lecture 5. Basic and Advanced SQL

(Chapter 6 and 7)

alisa.lincke@lnu.se

Outline

- SQL Data Definition and Data Types
- Specifying Constraints in SQL
- Basic Retrieval Queries in SQL
- INSERT, DELETE, and UPDATE Statements in SQL
- Break 10 min
- More Complex SQL Retrieval Queries:
 - JOIN, Aggregation, GROUP BY, nested queries

About SQL

- SQL (Structured Query Language, come from the word “SEQUEL” (Structured English Query Language)) is a standard language for storing, manipulating, and retrieving data in database.
- SQL was developed in the 1970s by IBM Computer Scientists
- Considered one of the major reasons for the commercial success of relational databases
- SQL uses the terms ***table***, ***row***, and ***column*** for the formal relational terms *relation*, *tuple*, and *attribute* respectively.
- SQL allows a table to have two or more tuples that are identical in all their attribute values
- Every statement in SQL finished by semicolon “;”
- SQL Syntax: <https://www.tutorialspoint.com/sql/sql-syntax.htm>

SQL Schema

- SQL Schema is a database description which :
 - Identified by a **schema name**
 - Includes **an authorization identifier** to indicate the user or account who owns schema
 - Includes description of each elements of the schema (tables, types, constraints, views, domains, etc.)
- A Schema is created via the CREATE SCHEMA statement which can include all the schema elements definitions.

Defining COMPANY Schema (1)

Some foreign keys may cause errors

Specified either via:

[Circular references](#)

Or because they refer to a
[table that has not yet been
created](#)

```
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),
  Super_ssn      CHAR(9),
  Dno            INT                  NOT NULL,
  PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
( Dname          VARCHAR(15)          NOT NULL,
  Dnumber        INT                  NOT NULL,
  Mgr_ssn        CHAR(9)              NOT NULL,
  Mgr_start_date DATE,
  PRIMARY KEY (Dnumber),
  UNIQUE (Dname),
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
( Dnumber        INT                  NOT NULL,
  Dlocation      VARCHAR(15)          NOT NULL,
  PRIMARY KEY (Dnumber, Dlocation),
  FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

Defining COMPANY Schema (1)

```
CREATE TABLE PROJECT
  ( Pname                VARCHAR(15)          NOT NULL,
    Pnumber              INT                  NOT NULL,
    Plocation            VARCHAR(15),
    Dnum                 INT                  NOT NULL,
    PRIMARY KEY (Pnumber),
    UNIQUE (Pname),
    FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE WORKS_ON
  ( Essn                 CHAR(9)              NOT NULL,
    Pno                  INT                  NOT NULL,
    Hours                DECIMAL(3,1)         NOT NULL,
    PRIMARY KEY (Essn, Pno),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
    FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );

CREATE TABLE DEPENDENT
  ( Essn                 CHAR(9)              NOT NULL,
    Dependent_name        VARCHAR(15)         NOT NULL,
    Sex                   CHAR,
    Bdate                 DATE,
    Relationship            VARCHAR(8),
    PRIMARY KEY (Essn, Dependent_name),
    FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Attribute Data Types in SQL

- **Basic data types**

- **Numeric data types**

- Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
 - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`

- **Character-string data types**

- Fixed length: `CHAR (n)`
 - Varying length: `VARCHAR (n)`

- **Bit-string data types**

- Fixed length: `BIT (n)`
 - Varying length: `BIT VARYING (n)`

- **Boolean data type**

- Values of `TRUE` or `FALSE` or `NULL`

- **DATE data type**

- Ten positions
 - Components are `YEAR`, `MONTH`, and `DAY` in the form `YYYY-MM-DD`
 - Multiple mapping functions available in RDBMSs to change date formats

Additional Data Types in SQL

- **Timestamp** data type

Includes the `DATE` and `TIME` fields

- Plus a minimum of six positions for decimal fractions of seconds
- Optional `WITH TIME ZONE` qualifier

- **INTERVAL** data type

- Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

- **DATE, TIME, Timestamp, INTERVAL** data types can be **cast** or converted to string formats for comparison

- Different DBMS have added more data types to SQL which are not presented in this lecture.

Domains in SQL

- **Domain**

- Name used with the attribute specification
- Makes it easier to change the data type for a domain that is used by numerous attributes
- Improves schema readability
- Example:
 - `CREATE DOMAIN SSN_TYPE AS CHAR(9);`

Specifying Constraints in SQL

- **Key constraint**: A primary key value cannot be duplicated
- **Entity Integrity Constraint**: A primary key value cannot be NULL
- **Referential integrity constraints** : The “foreign key “ must have a value that is already present as a primary key, or may be NULL.
- Restrictions on **attribute domains**:
 - Default value of an attribute
 - DEFAULT <value>
 - NULL is not permitted for a particular attribute (NOT NULL)
 - CHECK clause
 - Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

Specifying Key and Referential Integrity Constraints

PRIMARY KEY clause

Specifies one or more attributes that make up the primary key of a relation

Example: Dnumber INT PRIMARY KEY;

UNIQUE clause

Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).

Example: Dname VARCHAR(15) UNIQUE;

FOREIGN KEY clause

Default operation: reject update on violation

Attach referential triggered action clause:

Options include SET NULL, CASCADE, and SET DEFAULT

Action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE

CASCADE option suitable for “relationship” relations

Example

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

```
CREATE TABLE EMPLOYEE
( ... ,
  Dno          INT          NOT NULL      DEFAULT 1,
  CONSTRAINT EMPPK
    PRIMARY KEY (Ssn),
  CONSTRAINT EMPSUPERFK
    FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
      ON DELETE SET NULL      ON UPDATE CASCADE,
  CONSTRAINT EMPDEPTFK
    FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
      ON DELETE SET DEFAULT   ON UPDATE CASCADE);

CREATE TABLE DEPARTMENT
( ... ,
  Mgr_ssn CHAR(9)          NOT NULL      DEFAULT '888665555',
  ... ,
  CONSTRAINT DEPTPK
    PRIMARY KEY(Dnumber),
  CONSTRAINT DEPTSK
    UNIQUE (Dname),
  CONSTRAINT DEPTMGRFK
    FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
      ON DELETE SET DEFAULT   ON UPDATE CASCADE);

CREATE TABLE DEPT_LOCATIONS
( ... ,
  PRIMARY KEY (Dnumber, Dlocation),
  FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
      ON DELETE CASCADE      ON UPDATE CASCADE);
```

Basic Commands in SQL

- CREATE

- DROP

Used for creating/removing new database, table, column

- SELECT

- INSERT

- DELETE

- UPDATE

Used for manipulating with data in database

SELECT Command

- This command helps in retrieving a single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.
- Basic form of the SELECT statement:

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

where

- <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.

Selecting Records from a Single Table

- Retrieving all rows and columns from a table

```
SELECT *  
FROM company.employee;
```

- Retrieving a subset of rows from a table

```
SELECT fname,lname  
FROM company.employee;
```

- Retrieving a subset of rows with a condition

```
SELECT Bdate, Address  
FROM EMPLOYEE  
WHERE Fname='John' AND Minit='B' AND Lname='Smith';
```

Selecting Records from a Single Table

- **Sorting Query Results**

```
SELECT Fname,Lname  
FROM EMPLOYEE  
WHERE Dno= 5  
ORDER BY Salary ASC;
```

- **Searching for a particular substring or pattern.**

```
SELECT *  
FROM EMPLOYEE  
WHERE Address LIKE '%Houston%';
```

- **Retrieving distinct salary**

```
SELECT DISTINCT Salary  
FROM employee  
ORDER BY Salary ASC;
```


WHERE clause

- **Selection condition:**
 - **OR, AND:** Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions when multiple relations are involved.
- **Logical comparison operators**
=, <, <=, >, >=, and <>
- **Projection attributes**
 - Attributes whose values are to be retrieved
- **LIKE** comparison operator
 - Used for string **pattern matching**
 - % replaces an arbitrary number of zero or more characters
 - underscore (_) replaces a single character

Examples: **WHERE** Address **LIKE** '%Houston,TX%';
 WHERE Ssn **LIKE** '__ 1__ 8901';
- **BETWEEN** comparison operator
Example: **WHERE** (Salary **BETWEEN** 30000 **AND** 40000) **AND** Dno = 5;

Selecting data from multiple tables(1)

| <u>Bdate</u> | <u>Address</u> |
|--------------|--------------------------|
| 1965-01-09 | 731 Fondren, Houston, TX |

| <u>Fname</u> | <u>Lname</u> | <u>Address</u> |
|--------------|--------------|--------------------------|
| John | Smith | 731 Fondren, Houston, TX |
| Franklin | Wong | 638 Voss, Houston, TX |
| Ramesh | Narayan | 975 Fire Oak, Humble, TX |
| Joyce | English | 5631 Rice, Houston, TX |

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

Q1: **SELECT** Fname, Lname, Address
 FROM EMPLOYEE, DEPARTMENT
 WHERE Dname='Research' **AND** Dnumber=Dno;

Retrieving data from multiple tables (2)

(c)

| <u>Pnumber</u> | <u>Dnum</u> | <u>Lname</u> | <u>Address</u> | <u>Bdate</u> |
|----------------|-------------|--------------|------------------------|--------------|
| 10 | 4 | Wallace | 291Berry, Bellaire, TX | 1941-06-20 |
| 30 | 4 | Wallace | 291Berry, Bellaire, TX | 1941-06-20 |

Query 2. 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.

Q2: **SELECT** Pnumber, Dnum, Lname, Address, Bdate
 FROM PROJECT, DEPARTMENT, EMPLOYEE
 WHERE Dnum=Dnumber **AND** Mgr_ssn=Ssn **AND**
 Plocation='Stafford';

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes in different relations
 - As long as the attributes are in different relations
 - Must **qualify** the attribute name with the relation name to prevent ambiguity

```
Q1A:  SELECT  Fname, EMPLOYEE.Name, Address
        FROM    EMPLOYEE, DEPARTMENT
        WHERE   DEPARTMENT.Name='Research' AND
                DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, and Renaming

- **Aliases or tuple variables**

- Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query Example. 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;
```

- Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

Unspecified WHERE Clause and Use of the Asterisk

- Specify an asterisk (*)
 - Retrieve all the attribute values of the selected tuples
 - The * can be prefixed by the relation name; e.g., EMPLOYEE *

```
Q1C:  SELECT  *
      FROM    EMPLOYEE
      WHERE   Dno=5;

Q1D:  SELECT  *
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   Dname='Research' AND Dno=Dnumber;

Q10A: SELECT  *
      FROM    EMPLOYEE, DEPARTMENT;
```

DISTINCT clause

- SQL does not automatically eliminate duplicate tuples in query results
- For aggregate operations (See sec 7.1.7) duplicates must be accounted for
- Use the keyword **DISTINCT** in the `SELECT` clause
 - Only distinct tuples should remain in the result

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

Q11: **SELECT** **ALL** Salary
 FROM EMPLOYEE;

Q11A: **SELECT** **DISTINCT** Salary
 FROM EMPLOYEE;

Arithmetic Operations in SELECT

- **Standard arithmetic operators:**

- Addition (+), subtraction (−), multiplication (*), and division (/) may be included as a part of **SELECT**

Query Example: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal  
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P  
WHERE E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='ProductX';
```

- Also **standard statistical operations** (such as max, min, count, sum, mean, etc.)

Ordering of Query Results

- Use **ORDER BY** clause
 - Keyword **DESC** to see result in a descending order of values
 - Keyword **ASC** to specify ascending order explicitly
 - Typically placed at the end of the query

```
ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC
```

Break 10 min

Basic SQL Retrieval Query Block

```
SELECT    <attribute list>  
FROM      <table list>  
[ WHERE    <condition> ]  
[ ORDER BY <attribute list> ];
```

where, [] brackets mean not mandatory/optional
Every block is finished by semicolon ‘;’

INSERT Command

- **INSERT** typically inserts a tuple (row) in a relation (table)
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command
- Constraints on data types are observed automatically
- Any integrity constraints as a part of the DDL specification are enforced
- Specify the relation name and a list of values for the tuple. All values including nulls are supplied.

- Example:

```
U1:  INSERT INTO  EMPLOYEE
      VALUES      ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                    Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

DELETE

- Removes tuples from a relation
 - Includes a WHERE-clause to select the tuples to be deleted
 - Referential integrity should be enforced
 - 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

Example DELETE Command

- Removes tuples from a relation
 - Includes a `WHERE` clause to select the tuples to be deleted. The number of tuples deleted will vary.

| | | |
|-------------|--------------------|-------------------------|
| U4A: | DELETE FROM | EMPLOYEE |
| | WHERE | Lname='Brown'; |
| U4B: | DELETE FROM | EMPLOYEE |
| | WHERE | Ssn='123456789'; |
| U4C: | DELETE FROM | EMPLOYEE |
| | WHERE | Dno=5; |
| U4D: | DELETE FROM | EMPLOYEE; |

UPDATE (1)

- 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 specified as part of DDL specification is enforced
- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

UPDATE
SET
WHERE

PROJECT
PLOCATION = 'Bellaire', DNUM = 5
PNUMBER=10

UPDATE (2)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
UPDATE      EMPLOYEE
SET         SALARY = SALARY *1.1
WHERE DNO IN (SELECT DNUMBER
                FROM   DEPARTMENT
                WHERE  DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
 - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
 - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
 - nested queries,
 - joined tables and JOIN operations
 - aggregate functions, and grouping

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
 - **Unknown value**
 - **Unavailable or withheld value**
 - **Not applicable attribute**
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
 - TRUE, FALSE, and UNKNOWN (like Maybe)
- **NULL = NULL comparison is avoided**

Comparisons Involving NULL and Three-Valued Logic (cont'd.)

Table 7.1 Logical Connectives in Three-Valued Logic

| | | | | |
|-----|------------|---------|---------|---------|
| (a) | AND | TRUE | FALSE | UNKNOWN |
| | TRUE | TRUE | FALSE | UNKNOWN |
| | FALSE | FALSE | FALSE | FALSE |
| | UNKNOWN | UNKNOWN | FALSE | UNKNOWN |
| (b) | OR | TRUE | FALSE | UNKNOWN |
| | TRUE | TRUE | TRUE | TRUE |
| | FALSE | TRUE | FALSE | UNKNOWN |
| | UNKNOWN | TRUE | UNKNOWN | UNKNOWN |
| (c) | NOT | | | |
| | TRUE | FALSE | | |
| | FALSE | TRUE | | |
| | UNKNOWN | UNKNOWN | | |

Comparisons Involving NULL and Three-Valued Logic

- SQL allows queries that check whether an attribute value is NULL
 - IS or IS NOT NULL

Example:

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

Nested Queries using IN

Query: Make a list of all project numbers for projects that involved an employee whose last name is 'Smith' either as a worker or as a manager of the department that controls the project.

The first nested query selects the project numbers of projects that have an employee with last name 'Smith' involved as manager. The second nested query selects the project numbers of projects that have an employee with last name 'Smith' involved as worker.

```
Q4A:  SELECT  DISTINCT Pnumber
      FROM    PROJECT
      WHERE   Pnumber IN
            ( SELECT  Pnumber
              FROM    PROJECT, DEPARTMENT, EMPLOYEE
              WHERE   Dnum=Dnumber AND
                    Mgr_ssn=Ssn AND Lname='Smith' )

      OR

      Pnumber IN
            ( SELECT  Pno
              FROM    WORKS_ON, EMPLOYEE
              WHERE   Essn=Ssn AND Lname='Smith' );
```

Nested Queries using Comparison Operator

- Use other comparison operators to compare a single value v
 - `= ANY` (or `= SOME`) operator
 - Returns `TRUE` if the value v is equal to some value in the set V and is hence equivalent to `IN`
 - Other operators that can be combined with `ANY` (or `SOME`): `>`, `>=`, `<`, `<=`, and `<>`
 - `ALL`: value must exceed all values from nested query

```
SELECT  Lname, Fname
FROM    EMPLOYEE
WHERE   Salary > ALL ( SELECT Salary
                        FROM    EMPLOYEE
                        WHERE   Dno=5 );
```

Nested Queries using EXISTS

Retrieve the names of employees who have no dependents:

```
Select e.fname, e.lname  
from employee e  
where not EXISTS (select * from dependent d where e.ssn=d.essn);
```

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

```
select fname, lname  
from employee e  
where exists (select * from dependent as d where e.ssn=d.essn)  
and exists (select * from department as d2 where e.ssn=d2.mgrssn);
```

SQL Joins

LEFT JOIN



Everything on the left
+
anything on the right that
matches

```
SELECT *  
FROM TABLE_1  
LEFT JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI LEFT JOIN



Everything on the left
that is NOT on the right

```
SELECT *  
FROM TABLE_1  
LEFT JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY  
WHERE TABLE_2.KEY IS NULL
```

RIGHT JOIN



Everything on the right
+
anything on the left that matches

```
SELECT *  
FROM TABLE_1  
RIGHT JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI RIGHT JOIN



Everything on the right
that is NOT on the left

```
SELECT *  
FROM TABLE_1  
RIGHT JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY  
WHERE TABLE_1.KEY IS NULL
```

OUTER JOIN



Everything on the right
+
Everything on the left

```
SELECT *  
FROM TABLE_1  
OUTER JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI OUTER JOIN



Everything on the left and right
that is unique to each side

```
SELECT *  
FROM TABLE_1  
OUTER JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY  
WHERE TABLE_1.KEY IS NULL  
OR TABLE_2.KEY IS NULL
```

INNER JOIN



Only the things that match on the
left AND the right

```
SELECT *  
FROM TABLE_1  
INNER JOIN TABLE_2  
ON TABLE_1.KEY = TABLE_2.KEY
```

CROSS JOIN



All combination of rows from the
right and the left (cartesian
product)

```
SELECT *  
FROM TABLE_1  
CROSS JOIN TABLE_2
```


Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
 - **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Following query returns a single row of computed values from EMPLOYEE table:

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

- The result can be presented with new names:

```
SELECT SUM (Salary) AS Total_Sal, MAX (Salary) AS Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG (Salary) AS
Average_Sal
FROM EMPLOYEE;
```

Aggregate Functions in SQL (cont'd.)

NULL values are discarded when aggregate functions are applied to a particular column

Query 20. 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.

```
Q20:  SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT    COUNT (*)
      FROM      EMPLOYEE;
```

```
Q22:  SELECT    COUNT (*)
      FROM      EMPLOYEE, DEPARTMENT
      WHERE     DNO=DNUMBER AND DNAME='Research';
```

GROUP BY clause

- The grouping attribute must appear in the SELECT clause:

Example: **SELECT** Dno, **COUNT** (*), **AVG** (Salary)
 FROM EMPLOYEE
 GROUP BY Dno;

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

Example: **SELECT** Pnumber, Pname, **COUNT** (*)
 FROM PROJECT, WORKS_ON
 WHERE Pnumber=Pno
 GROUP BY Pnumber, Pname;

Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]  
                             { , <column name> <column type> [ <attribute constraint> ] }  
                             [ <table constraint> { , <table constraint> } ] )
```

```
DROP TABLE <table name>  
ALTER TABLE <table name> ADD <column name> <column type>
```

```
SELECT [ DISTINCT ] <attribute list>  
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }  
[ WHERE <condition> ]  
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]  
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]
```

```
<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )  
                    { , ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) ) } ) )
```

```
<grouping attributes> ::= <column name> { , <column name> }
```

```
<order> ::= ( ASC | DESC )
```

```
INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]  
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }  
| <select statement> )
```

Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

DELETE FROM <table name>

[WHERE <selection condition>]

UPDATE <table name>

SET <column name> = <value expression> { , <column name> = <value expression> }

[WHERE <selection condition>]

CREATE [UNIQUE] INDEX <index name>

ON <table name> (<column name> [<order>] { , <column name> [<order>] })

[CLUSTER]

DROP INDEX <index name>

CREATE VIEW <view name> [(<column name> { , <column name> })]

AS <select statement>

DROP VIEW <view name>

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

More Examples from the Book

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT  E.Fname, E.Lname
      FROM    EMPLOYEE AS E
      WHERE   E.Ssn IN ( SELECT  D.Essn
                        FROM    DEPENDENT AS D
                        WHERE   E.Fname = D.Dependent_name
                        AND E.Sex = D.Sex );
```

Query 6. Retrieve the names of employees who have no dependents.

```
Q6:   SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   NOT EXISTS ( SELECT  *
                        FROM    DEPENDENT
                        WHERE   Ssn = Essn );
```

Query 7. List the names of managers who have at least one dependent.

```
Q7:   SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   EXISTS ( SELECT  *
                        FROM    DEPENDENT
                        WHERE   Ssn = Essn )
      AND
      EXISTS ( SELECT  *
                        FROM    DEPARTMENT
                        WHERE   Ssn = Mgr_ssn );
```

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

Query 17. Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

```
Q17:  SELECT  DISTINCT Essn
      FROM    WORKS_ON
      WHERE   Pno IN (1, 2, 3);
```


Query 19. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
Q19:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    EMPLOYEE;
```

Query 20. 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.

```
Q20:  SELECT  SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM    (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
      WHERE   Dname = 'Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT  COUNT (*)
      FROM    EMPLOYEE;

Q22:  SELECT  COUNT (*)
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   DNO = DNUMBER AND DNAME = 'Research';
```

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:  SELECT  Dno, COUNT (*), AVG (Salary)
      FROM    EMPLOYEE
      GROUP BY Dno;
```

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
Q25:  SELECT  Pnumber, Pname, COUNT (*)
      FROM    PROJECT, WORKS_ON
      WHERE   Pnumber = Pno
      GROUP BY Pnumber, Pname;
```

Query 26. 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.

```
Q26:  SELECT  Pnumber, Pname, COUNT (*)
      FROM    PROJECT, WORKS_ON
      WHERE   Pnumber = Pno
      GROUP BY Pnumber, Pname
      HAVING  COUNT (*) > 2;
```

Query 27. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
SELECT Pnumber, Pname, COUNT (*)
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber = Pno AND Ssn = Essn AND Dno = 5
GROUP BY Pnumber, Pname;
```

Query 13. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10% raise.

```
Q13: SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND
P.Pname = 'ProductX';
```

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
Q12: SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston,TX%';
```

Query 12A. Find all employees who were born during the 1950s.

```
Q12: SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '__7____';
```


Preparation for Exam

- Basic queries:
 - Selecting data from one table
 - Selecting data from several tables
 - Order By
 - Count(), SUM() operations
- Advanced queries:
 - Nested queries
 - Group By