Basic SQL Queries

- Elmisri Navathe, 6th Edition

SQL Data Definition and Data Types

- SQL uses the terms **table**, **row**, and **column** for the formal relational model terms *relation*, *tuple*, and *attribute*, respectively.
- The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), and domains (as well as other constructs such as views, assertions, and triggers).
- **4.1.1 Schema and Catalog Concepts in SQL**
- **4.1.2 The CREATE TABLE Command in SQL**
- **4.1.3 Attribute Data Types and Domains in SQL**

4.1.1 Schema and Catalog Concepts in SQL

- An SQL schema is identified by a **schema name**, and 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 **tables, constraints, views, domains**, and other constructs (such as **authorization** grants) that describe the schema.
- A schema is created via the **CREATE SCHEMA** statement, which can include all the schema elements definitions or the schema can be assigned a name and authorization identifier, and the **elements can be defined later.**
- For example, the following statement creates a schema called COMPANY, owned by the user with authorization identifier 'Jsmith'.
- Note that each statement in **SQL ends with a semicolon**.
- **CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';**
- The **privilege** to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the **system administrator or DBA**.

4.1.1 Schema and Catalog Concepts in SQL

- SQL uses the concept of a **catalog**—a **named collection of schemas** in an SQL environment.
- An **SQL environment** is basically an installation of an **SQL-compliant RDBMS** on a computer system.
- A catalog always contains a special schema called **INFORMATION_SCHEMA,** which provides **information on all the schemas** in the catalog **and all the element descriptors** in these schemas.
- **Referential integrity** can be defined between relations only if they exist in schemas **within the same catalog.**
- Schemas within the same catalog can also **share domain definitions**.

4.1.2 The CREATE TABLE Command in SQL

- The **CREATE TABLE** command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.

- The attributes are specified first, and each attribute is given a 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 the attributes are declared, or they can be added later using the ALTER TABLE command

- Alternatively, we can explicitly attach the schema name to the relation name, separated by a period. For example, by writing

- CREATE TABLE COMPANY.EMPLOYEE ... rather than CREATE TABLE EMPLOYEE ...

4.1.2 The CREATE TABLE Command in SQL

```
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),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

4.1.2 The CREATE TABLE Command in SQL

- The relations declared through CREATE TABLE statements are called **base tables** (or base relations); this means that the relation and its tuples are actually created and stored as a file by the DBMS.
- Base relations are distinguished from **virtual relations**, created through the **CREATE VIEW** statement, which may or may not correspond to an actual physical file.
- In SQL, the **attributes** in a base table are **considered to be ordered** in the sequence in which they are specified in the CREATE TABLE statement.
- However, rows **(tuples) are not considered to be ordered** within a relation.

4.1.2 The CREATE TABLE Command in SQL

- There are some foreign keys that may cause errors because they are specified either via **circular references** or because they **refer to a table that has not yet been created.**
- For example, the foreign key Super_ssn in the EMPLOYEE table is a circular reference because it refers to the table itself.
- The foreign key Dno in the EMPLOYEE table refers to the DEPARTMENT table, which has not been created yet.
- To deal with this type of problem, these constraints can be left out of the initial CREATE TABLE statement, and then **added later using the ALTER TABLE statement.**

4.1.3 Attribute Data Types and Domains in SQL

- The basic data types available for attributes include numeric, character string, bit string, Boolean, date, and time.

■ Numeric data types include integer numbers of various sizes **(INTEGER or INT, and SMALLINT)** and floating-point (real) numbers of various precision (**FLOAT or REAL, and DOUBLE PRECISION**). Formatted numbers can be declared by using **DECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)**—where **i, the precision, is the total number of decimal digits** and **j, the scale, is the number of digits after the decimal point**.

■ Character

- string data types are either fixed length—**CHAR(n) or CHARACTER(n),** where n is the number of characters—or varying length— **VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n),** where n is the maximum number of characters. When specifying a literal string value, it is placed between **single quotation marks** (apostrophes), and it is **case sensitive** (a distinction is made between uppercase and lowercase).

4.1.3 Attribute Data Types and Domains in SQL

- For **fixed length strings**, a shorter **string is padded with blank characters** to the right. For example, if the value ‘Smith’ is for an attribute of type **CHAR(10),** it is padded with five blank characters to become ‘Smith ’ if needed. Padded blanks are generally **ignored when strings are compared**.

- For comparison purposes, strings are considered ordered in alphabetic (or lexicographic) order; if a string **str1 appears before another string str2 in alphabetic order, then str1 is considered to be less than str2.**

- There is also a **concatenation operator denoted by ||** (double vertical bar) that can concatenate two strings in SQL. For example, **‘abc’ || ‘XYZ’** results in a single string **‘abcXYZ’.**

- Another variable-length string data type called **CHARACTER LARGE OBJECT or CLOB** is also available to specify columns that have large text values, such as documents. The CLOB maximum length can be specified in **kilobytes (K), megabytes (M), or gigabytes (G).** For example, CLOB (20M) specifies a maximum length of 20 megabytes.

4.1.3 Attribute Data Types and Domains in SQL

■ Bit-string data types are either of fixed length n—**BIT(n)**—or varying length—**BIT VARYING(n),** where n is the maximum number of bits.

- The **default** for n, the **length** of a character string or bit string, is **1**. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, **B‘10101’**

- Another variable-length bitstring data type called **BINARY LARGE OBJECT or BLOB** is also available to specify columns that have **large binary values**, such as **images**.

- The maximum length of a BLOB can be specified in **kilobits (K), megabits (M), or gigabits (G)**. For example, BLOB(30G) specifies a maximum length of 30 gigabits.

4.1.3 Attribute Data Types and Domains in SQL

■ A Boolean data type has the traditional values of **TRUE or FALSE**. In SQL, because of the presence of **NULL values, a three-valued logic** is used, so a third possible value for a Boolean data type is **UNKNOWN**.

■ The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form **YYYY-MM-DD**.

- The **TIME** data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form **HH:MM:SS**.
- Months should be between 1 and 12 and dates must be between 1 and 31; furthermore, a date should be a valid date for the corresponding month.

4.1.3 Attribute Data Types and Domains in SQL

- The **< (less than) comparison** can be used with dates or times—an earlier date is considered to be smaller than a later date, and similarly with time.
- Literal values are represented by single-quoted strings preceded by the keyword DATE or TIME; for example, DATE ‘2008-09- 27’ or TIME ‘09:12:47’.
- In addition, a data type **TIME(i)**, where i is called time fractional seconds precision, specifies i + 1 additional positions for TIME—one position for an additional period (.) separator character, and i positions for specifying decimal fractions of a second.
- A **TIME WITH TIME ZONE** data type includes an additional six positions for specifying the **displacement from the standard universal time zone**, which is in the range +13:00 to –12:59 in units of HOURS:MINUTES. If WITH TIME ZONE is not included, the default is the local time zone for the SQL session.

4.1.3 Attribute Data Types and Domains in SQL

- ■ A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.
- ■ Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type.
- This specifies an interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp.
- Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals.
- The format of DATE, TIME, and TIMESTAMP can be considered as a special type of string.
- Hence, they can generally be used in string comparisons by being cast (or coerced or converted) into the equivalent strings.
- It is possible to specify the data type of each attribute directly, or a domain can be declared, and the domain name used with the attribute specification.

4.1.3 Attribute Data Types and Domains in SQL

- This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability.
- For example, we can create a domain SSN_TYPE by the following statement:
- **CREATE DOMAIN SSN_TYPE AS CHAR(9);**
- We can use SSN_TYPE in place of CHAR(9) for the attributes Ssn and Super_ssn of EMPLOYEE, Mgr_ssn of DEPARTMENT, Essn of WORKS_ON, and Essn of DEPENDENT.
- A domain can also have an optional default specification via a DEFAULT clause.

4.2 Specifying Constraints in SQL

- The basic constraints that can be specified in SQL as part of table creation include key and referential integrity constraints, restrictions on attribute domains and NULLs, and constraints on individual tuples within a relation.
- **4.2.1 Specifying Attribute Constraints and Attribute Defaults**
- **4.2.2 Specifying Key and Referential Integrity Constraints**
- **4.2.3 Giving Names to Constraints**
- **4.2.4 Specifying Constraints on Tuples Using CHECK**

4.2.1 Specifying Attribute Constraints and Attribute Defaults

- A *constraint* NOT NULL may be specified if NULL is not permitted for a particular attribute.
- This is always implicitly specified for *primary key* and explicitly for any other.
- A *default value* for an attribute can be defined by appending the clause **DEFAULT** <value> to an attribute definition. The default value is included in any new tuple if an explicit value is not provided for that attribute.
- Figure 4.2 illustrates an example of specifying a default manager for a new department and a default department for a new employee.
- If no default clause is specified, the default *default value* is NULL for attributes *that do not have* the NOT NULL constraint.

4.2.1 Specifying Attribute Constraints and Attribute Defaults

```
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);
```

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

4.2.1 Specifying Attribute Constraints and Attribute Defaults

- Another type of constraint can restrict attribute or domain values using the **CHECK** clause following an attribute or domain definition.
- For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:
- Dnumber INT **NOT NULL CHECK** (Dnumber > 0 **AND** Dnumber < 21);
- The CHECK clause can also be used in conjunction with the CREATE DOMAIN statement.
- For example, we can write the following statement:
- **CREATE DOMAIN** D_NUM **AS** INTEGER **CHECK** (D_NUM > 0 **AND** D_NUM < 21);
- We can then use the created domain D_NUM as the attribute type for all attributes that refer to department numbers, such as Dnumber of DEPARTMENT, Dnum of PROJECT, Dno of EMPLOYEE, and so on.

4.2.2 Specifying Key and Referential Integrity Constraints

- The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a *single* attribute, the clause can follow the attribute directly. For example, the primary key of DEPARTMENT can be specified as follows:

- Dnumber INT **PRIMARY KEY**;

- The **UNIQUE** clause specifies alternate (secondary) keys.

- The **UNIQUE** clause can also be specified directly for a secondary key if the secondary key is a single attribute, as in the following example:

- Dname VARCHAR(15) **UNIQUE**;

- Referential integrity is specified via the **FOREIGN KEY** clause,

- A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified.

- The default action that SQL takes for an integrity violation is to **reject** the update operation that will cause a violation, which is known as the RESTRICT option.

- However, the schema designer can specify an alternative action to be taken by attaching a **referential triggered action** clause to any foreign key constraint.

4.2.2 Specifying Key and Referential Integrity Constraints

- The options include SET NULL, CASCADE, and SET DEFAULT.
- An option must be qualified with either ON DELETE or ON UPDATE.
- ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE.
- This means that if the tuple for a *supervising employee* is *deleted*, the value of Super_ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple.
- On the other hand, if the Ssn value for a supervising employee is *updated* (say, because it was entered incorrectly), the new value is *cascaded* to Super_ssn for all employee tuples referencing the updated employee tuple.
- In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE.
- 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 referencing foreign key attribute to the updated (new) primary key value for all the referencing tuples.
- It is the responsibility of the database designer to choose the appropriate action and to specify it in the database schema.
- As a general rule, the CASCADE option is suitable for “relationship” relations, such as WORKS_ON; for relations that represent multivalued attributes, such as DEPT_LOCATIONS; and for relations that represent weak entity types, such as DEPENDENT.

4.2.3 Giving Names to Constraints

- A constraint may be given a **constraint name**, following the keyword **CONSTRAINT**.
- The names of all constraints within a particular schema must be unique.
- A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint.
- Giving names to constraints is optional.

4.2.4 Specifying Constraints on Tuples Using CHECK

- Other *table constraints* can be specified through additional CHECK clauses at the end of a CREATE TABLE statement.
- These can be called **tuple-based** constraints because they apply to each tuple *individually* and are checked whenever a tuple is inserted or modified.
- For example, suppose that the DEPARTMENT table had an additional attribute Dept_create_date, which stores the date when the department was created. Then we could add the following CHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date.
- **CHECK** (Dept_create_date <= Mgr_start_date);

4.3 Basic Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database: the **SELECT** statement.
- **4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries**
- **4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables**
- **4.3.3 Unspecified WHERE Clause and Use of the Asterisk**
- **4.3.4 Tables as Sets in SQL**
- **4.3.5 Substring Pattern Matching and Arithmetic Operators**
- **4.3.6 Ordering of Query Results**
- **4.3.7 Discussion and Summary**
- **of Basic SQL Retrieval Queries**

4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

- The basic form of the SELECT statement, sometimes called a **mapping** or a **select-from-where block**, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

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.
- In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and <>.

4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

- **Query 0.** Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

Q0: SELECT Bdate, Address

FROM EMPLOYEE

WHERE Fname='John' **AND** Minit='B' **AND** Lname='Smith';

- The SELECT clause of SQL specifies the attributes whose values are to be retrieved, which are called the **projection attributes**, and the WHERE clause specifies the Boolean condition that must be true for any retrieved tuple, which is known as the **selection condition**.

4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

- **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;

- In the WHERE clause of Q1, the condition Dname = 'Research' is a **selection condition** that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT. The condition Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE.

4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

- In general, any number of selection and join conditions may be specified in a single SQL query.
- A query that involves only selection and join conditions plus projection attributes is known as a **select-project-join** query.
- **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';

- The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple, whereas the join condition Mgr_ssn = Ssn relates the controlling department tuple to the employee tuple who manages that department. Each tuple in the result will be a *combination* of one project, one department, and one employee that satisfies the join conditions. The projection attributes are used to choose the attributes to be displayed from each combined tuple.

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- 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 multitable 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.

Q1A: SELECT Fname, EMPLOYEE.Name, Address

FROM EMPLOYEE, DEPARTMENT

WHERE DEPARTMENT.Name='Research' **AND** DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- Fully qualified attribute names can be used for clarity even if there is no ambiguity in attribute names.

Q1_: SELECT EMPLOYEE.Fname, EMPLOYEE.LName,

EMPLOYEE.Address

FROM EMPLOYEE, DEPARTMENT

WHERE DEPARTMENT.DName='Research' **AND** DEPARTMENT.Dnumber=EMPLOYEE.Dno;

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice, as in the following example.

- We can also create an *alias* for each table name to avoid repeated typing of long table names

- **Query 8.** For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

Q8: SELECT E.Fname, E.Lname, S.Fname, S.Lname

FROM EMPLOYEE **AS** E, EMPLOYEE **AS** S

WHERE E.Super_ssn=S.Ssn;

- In this case, we are required to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation.

- An alias can follow the keyword **AS**, as shown in Q8, or it can directly follow the relation name—for example, by writing EMPLOYEE E, EMPLOYEE S in the FROM clause of Q8.

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- It is also possible to **rename** the relation attributes within the query in SQL by giving them aliases.
- For example, if we write EMPLOYEE **AS** E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno) in the FROM clause, Fn becomes an alias for Fname, Mi for Minit, Ln for Lname, and so on.
- We can use this alias-naming mechanism in any SQL query to specify tuple variables for every table in the WHERE clause, whether or not the same relation needs to be referenced more than once.

Q1B: SELECT E.Fname, E.LName, E.Address

FROM EMPLOYEE E, DEPARTMENT D

WHERE D.DName='Research' **AND** D.Dnumber=E.Dno;

4.3.3 Unspecified WHERE Clause and Use of the Asterisk

- A *missing* WHERE clause indicates no condition on tuple selection; hence, *all tuples* of the relation specified in the FROM clause qualify and are selected for the query result. For example, Query 9 selects all EMPLOYEE Ssns

Q9: SELECT Ssn

FROM EMPLOYEE;

4.3.3 Unspecified WHERE Clause and Use of the Asterisk

- If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT—*all possible tuple combinations*—of these relations is selected.
- Query 10 selects all combinations of an EMPLOYEE Ssn and a DEPARTMENT Dname, regardless of whether the employee works for the department or not.

Q10: SELECT Ssn, Dname

FROM EMPLOYEE, DEPARTMENT;

4.3.3 Unspecified WHERE Clause and Use of the Asterisk

- To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an *asterisk* (*), which stands for *all the attributes.*
- For example, query Q1C retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5

Q1C: SELECT *

FROM EMPLOYEE

WHERE Dno=5;

4.3.3 Unspecified WHERE Clause and Use of the Asterisk

- query Q1D retrieves all the attributes of an EMPLOYEE and the attributes of the DEPARTMENT in which he or she works for every employee of the ‘Research’ department

Q1D: SELECT *

FROM EMPLOYEE, DEPARTMENT

WHERE Dname=‘Research’ **AND** Dno=Dnumber;

- Q10A specifies the CROSS PRODUCT of the EMPLOYEE and DEPARTMENT relations.

Q10A: SELECT *

FROM EMPLOYEE, DEPARTMENT;

4.3.4 Tables as Sets in SQL

- SQL usually treats a table not as a set but rather as a **multiset**; *duplicate tuples can appear more than once* in a table, and in the result of a query.
- SQL does not automatically eliminate duplicate tuples in the results of queries, for the following reasons:
  - ■ Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
  - ■ The user may want to see duplicate tuples in the result of a query.
  - ■ When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates.
- An SQL table with a key is restricted to being a set, since the key value must be distinct in each tuple.
- If we do not want duplicate tuples from the result of an SQL query, we use the keyword DISTINCT in the SELECT clause, meaning that only distinct tuples should remain in the result.

4.3.4 Tables as Sets in SQL

- In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.
- Specifying SELECT with neither ALL nor DISTINCT—as in our previous examples— is equivalent to SELECT ALL.
- For example, Q11 retrieves the salary of every employee; if several employees have the same salary, that salary value will appear as many times in the result of the query.
- If we are interested only in distinct salary values, we want each value to appear only once, regardless of how many employees earn that salary.
- By using the keyword **DISTINCT** as in Q11A, we accomplish this.
- **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;

4.3.4 Tables as Sets in SQL

- SQL has directly incorporated some of the set operations from mathematical *settheory*.
- The relations resulting from these set operations are sets of tuples; that is, *duplicate tuples are eliminated from the result.*
- These set operations apply only to *union-compatible relations*, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.
- The first SELECT query retrieves the projects that involve a 'Smith' as manager of the department that controls the project, and the second retrieves the projects that involve a 'Smith' as a worker on the project.
- SQL also has corresponding multiset operations, which are followed by the keyword
- **ALL** (UNION ALL, EXCEPT ALL, INTERSECT ALL). Their results are multisets (duplicates are not eliminated).

4.3.4 Tables as Sets in SQL

Query 4. 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 the project.

Q4A: (**SELECT DISTINCT** Pnumber

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE Dnum=Dnumber **AND** Mgr_ssn=Ssn

AND Lname=‘Smith’)

UNION

(**SELECT DISTINCT** Pnumber

FROM PROJECT, WORKS_ON, EMPLOYEE

WHERE Pnumber=Pno **AND** Essn=Ssn

AND Lname=‘Smith’);

4.3.5 Substring Pattern Matching and Arithmetic Operators

- The first feature of SQL that allows comparison conditions on only parts of a character string, using the **LIKE** comparison operator.
- This can be used for string **pattern matching**.
- Partial strings are specified using two reserved characters: % replaces an arbitrary number of zero or more characters, and the underscore (_) replaces a single character.
- **Query 12.** Retrieve all employees whose address is in Houston, Texas.

Q12: SELECT Fname, Lname

FROM EMPLOYEE

WHERE Address **LIKE** '%Houston,TX%';

4.3.5 Substring Pattern Matching and Arithmetic Operators

- To retrieve all employees who were born during the 1950s, we can use Query Q12A.
- **Query 12A.** Find all employees who were born during the 1950s.

Q12: SELECT Fname, Lname

FROM EMPLOYEE

WHERE Bdate **LIKE** '_ _ 5 _ _ _ _ _ _ _ _';

- Here, '5'must be the third character of the string (according to the format for date),
- each underscore serving as a placeholder for an arbitrary character.

4.3.5 Substring Pattern Matching and Arithmetic Operators

- 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.
- For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF' because \ is specified as the escape character.
- Any character not used in the string can be chosen as the escape character.
- Also, we need a rule to specify apostrophes or single quotation marks (' ') if they are to be included in a string because they are used to begin and end strings.
- If an apostrophe (') is needed, it is represented as two consecutive apostrophes (") so that it will not be interpreted as ending the string.

4.3.5 Substring Pattern Matching and Arithmetic Operators

- Another feature allows the use of arithmetic in queries.
- The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.
- For example, suppose that we want to see the effect of giving all employees who work on the 'ProductX' project a 10 percent raise; we can issue Query 13 to see what their salaries would become.
- **Query 13.** Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent 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';

4.3.5 Substring Pattern Matching and Arithmetic Operators

- For string data types, the concatenate operator || can be used in a query to append two string values.
- For date, time, timestamp, and interval data types, operators include incrementing (+) or decrementing (–) a date, time, or timestamp by an interval.
- In addition, an interval value is the result of the difference between two date, time, or timestamp values.
- Another comparison operator, which can be used for convenience, is BETWEEN.
- **Query 14.** Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.
- **Q14: SELECT** *

FROM EMPLOYEE

WHERE (Salary **BETWEEN** 30000 **AND** 40000) **AND** Dno = 5;

- The condition (Salary BETWEEN 30000 AND 40000) in Q14 is equivalent to the condition

((Salary >= 30000) AND (Salary <= 40000)).

4.3.6 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 of the attributes that appear in the query result, by using the **ORDER BY** clause.
- **Query 15.** Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.
- **Q15: SELECT** D.Dname, E.Lname, E.Fname, P.Pname

FROM DEPARTMENT D, EMPLOYEE E, WORKS_ON W,

PROJECT P

WHERE D.Dnumber= E.Dno **AND** E.Ssn= W.Essn **AND** W.Pno= P.Pnumber

ORDER BY D.Dname, E.Lname, E.Fname;

4.3.6 Ordering of Query Results

- The default order is in ascending order of values.
- We can specify the keyword **DESC** if we want to see the result in a descending order of values.
- The keyword **ASC** can be used to specify ascending order explicitly.
- For example, if we want descending alphabetical order on Dname and ascending order on Lname, Fname, the ORDER BY clause of Q15 can be written as
- **ORDER BY** D.Dname **DESC**, E.Lname **ASC**, E.Fname **ASC**

4.4 INSERT, DELETE, and UPDATE Statements in SQL

- Three commands to modify database: INSERT, DELETE and UPDATE
- **4.4.1 The INSERT Command**
- **4.4.2 The DELETE Command**
- **4.4.3 The UPDATE Command**

4.4.1 The INSERT Command

- INSERT is used to add a single tuple to a relation.
- We must specify the relation name and a list of values for the tuple.
- The values should be listed *in the same order* in which the corresponding attributes were specified in the CREATE TABLE command.
- For example, to add a new tuple to the EMPLOYEE relation, we can use U1:
- **U1: INSERT INTO** EMPLOYEE **VALUES** (‘Richard’, ‘K’, ‘Marini’, ‘653298653’, ‘1962-12-30’, ’98 Oak Forest, Katy, TX’, ‘M’, 37000, ‘653298653’, 4);

4.4.1 The INSERT Command

- A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.
- However, the values must include all attributes with NOT NULL specification *and* no default value.
- Attributes with NULL allowed or DEFAULT values are the ones that can be *left out.*
- For example, to enter a tuple for a new EMPLOYEE for whom we know only the Fname, Lname, Dno, and Ssn attributes, we can use U1A:
- **U1A: INSERT INTO** EMPLOYEE (Fname, Lname, Dno, Ssn) **VALUES** (‘Richard’, ‘Marini’, 4, ‘653298653’);

4.4.1 The INSERT Command

- Attributes not specified in U1A are set to their DEFAULT or to NULL, and the values are listed in the same order as the *attributes are listed in the INSERT* command itself.
- It is also possible to insert into a relation *multiple tuples* separated by commas in a single INSERT command.
- The attribute values forming *each tuple* are enclosed in parentheses.
- A DBMS that fully implements SQL should support and enforce all the integrity constraints that can be specified in the DDL.
- **U3: INSERT INTO** EMPLOYEE (Fname, Lname, Ssn, Dno) **VALUES** (‘Robert’, ‘Hatcher’, ‘980760540’, 2);
- (U2 is rejected if referential integrity checking is provided by DBMS.)
- **U2A: INSERT INTO** EMPLOYEE (Fname, Lname, Dno) **VALUES** (‘Robert’, ‘Hatcher’, 5);
- (U2A is rejected if NOT NULL checking is provided by DBMS.)

4.4.1 The INSERT Command

- A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the *result of a query.*

- For example, to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project, we can write the statements in U3A and U3B:

- **U3A: CREATE TABLE** WORKS_ON_INFO (Emp_name VARCHAR(15), Proj_name VARCHAR(15), Hours_per_week DECIMAL(3,1));

- **U3B: INSERT INTO** WORKS_ON_INFO (Emp_name, Proj_name, Hours_per_week)

SELECT E.Lname, P.Pname, W.Hours

FROM PROJECT P, WORKS_ON W, EMPLOYEE E

WHERE P.Pnumber=W.Pno **AND** W.Essn=E.Ssn;

- WORKS_ON_INFO table may not be up-to-date; that is, if we update any of the PROJECT, WORKS_ON, or EMPLOYEE relations after issuing U3B

4.4.2 The DELETE Command

- The DELETE command removes tuples from a relation.
- It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time.
- However, the deletion may propagate to tuples in other relations if *referential triggered actions* are specified in the referential integrity constraints of the DDL.
- Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command.
- A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table. We must use the DROP TABLE command to remove the table definition.

4.4.2 The DELETE Command

- The DELETE commands in U4A to U4D, if applied independently to the database in Figure 3.6, will delete zero, one, four, and all tuples, respectively, from the EMPLOYEE relation:
- **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;

4.4.3 The UPDATE Command

- The **UPDATE** command is used to modify attribute values of one or more selected tuples.
- As in the DELETE command, a WHERE clause in the UPDATE command
- selects the tuples to be modified from a single relation.
- However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a *referential triggered action* is specified in the referential integrity constraints of the DDL.
- An additional **SET** clause in the UPDATE command specifies the attributes to be modified and their new values.
- For example, to change the location and controlling department number of project number 10 to ‘Bellaire’ and 5, respectively, we use U5:

4.4.3 The UPDATE Command

- **U5: UPDATE** PROJECT **SET** Plocation = ‘Bellaire’, Dnum = 5 **WHERE** Pnumber=10;
- Several tuples can be modified with a single UPDATE command.
- An example is to give all employees in the ‘Research’ department a 10 percent raise in salary, as shown in U6.
- **U6: UPDATE** EMPLOYEE **SET** Salary = Salary * 1.1 **WHERE** Dno = 5;
- It is also possible to specify NULL or DEFAULT as the new attribute value.Notice that each UPDATE command explicitly refers to a single relation only.
- To modify multiple relations, we must issue several UPDATE commands.

4.5 Additional Features of SQL

■ SQL features: various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins, and recursive queries; SQL views, triggers, and assertions and commands for schema modification.

■ SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more databases. These include embedded (and dynamic) SQL, SQL/CLI (Call Level Interface) and its predecessor ODBC (Open Data Base Connectivity), and SQL/PSM (Persistent Stored Modules).

■ Each commercial RDBMS will have, in addition to the SQL commands, a set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes. We called these commands a *storage definition language* (*SDL*). Earlier versions of SQL had commands for **creating indexes**, but these were removed from the language because they were not at the conceptual schema level. Many systems still have the CREATE INDEX commands.

4.5 Additional Features of SQL

- SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.

- SQL has language constructs for specifying the *granting and revoking of privileges* to users. Privileges typically correspond to the right to use certain SQL commands to access certain relations. Each relation is assigned an owner, and either the owner or the DBA staff can grant to selected users the privilege to use an SQL statement—such as SELECT, INSERT, DELETE, or UPDATE—to access the relation. In addition, the DBA staff can grant the privileges to create schemas, tables, or views to certain users.

- SQL has language constructs for creating triggers. These are generally referred to as **active database** techniques, since they specify actions that are automatically triggered by events such as database updates.

4.5 Additional Features of SQL

■ SQL has incorporated many features from object-oriented models to have more powerful capabilities, leading to enhanced relational systems known as **object-relational**. Capabilities such as creating complex-structured attributes (also called **nested relations**), specifying abstract data types (called **UDT**s or user-defined types) for attributes and tables.

■ SQL and relational databases can interact with new technologies such as XML and OLAP.

END of chapter 4