Unit 6.1
Structured Query Language (SQL)

- Introduction
- Data Definition and Data Types
- Specifying Constraints
- Basic Retrieval Queries
- INSERT, DELETE, and UPDATE Statements
- Complex Retrieval Queries
- Views

Introduction

- SQL is considered one of the major reasons for the commercial success of relational databases.
- SQL includes some features from relational algebra and many features from tuple relational calculus. SQL syntax is more user-friendly than either of the two formal languages.
- Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. The name SQL is presently expanded as Structured Query Language.
- SQL is now the standard language for commercial relational DBMSs.

Introduction

- The standardization of SQL is a joint effort by the American National Standards Institute (ANSI) and the International Standards Organization (ISO), and the first SQL standard is called SQL-86 or SQL1.
- A revised and much expanded standard called SQL-92 (also referred to as SQL2) was subsequently developed.
- The next standard that is well-recognized is SQL:1999, which started out as SQL3.
- Additional updates to the standard are SQL:2003 and SQL:2006, which added XML features among other updates to the language.
- Another update in 2008 incorporated more object database features into SQL, and a further update is SQL:2011.

Introduction

- SQL has statements for data definitions, queries, and updates. It also has facilities for defining views, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It can also be embedded into programming languages.
- The later SQL standards (starting with SQL:1999) are divided into a core specification plus specialized extensions. Core is supposed to be implemented by all RDBMS vendors. The extensions can be implemented as optional modules to be purchased independently for specific database applications such as data mining, spatial data, temporal data, data warehousing, online analytical processing (OLAP), multimedia data, and so on.

Data Definition and Data Types

- SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively. These terms can be used interchangeably.
- The main SQL command for data definition is the **CREATE** statement, which can be used to create schemas, tables (relations), types, and domains, as well as other constructs such as views, assertions, and triggers.
- **Schema and Catalog Concepts in SQL:**
  - Early versions of SQL did not include the concept of schema; all tables (relations) were considered part of the same schema.
  - SQL2 introduced concept of schema to group together tables and other constructs that belong to the same database application (in some systems, a schema is called a database).

Data Definition and Data Types

- 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, types, 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. Alternatively, the schema can be assigned a name and authorization identifier, and 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';

Data Definition and Data Types

- In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.
- In addition to the concept of a schema, SQL uses the concept of a **catalog** – a named collection of schemas. Integrity constraints such as 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 certain elements, such as type and domain definitions.
- We can delete the schema using DROP SCHEMA command. For example,

  DROP SCHEMA COMPANY;

Data Definition and Data Types

- **The CREATE TABLE Command in SQL:**
  - CREATE TABLE command is used to specify 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 possibly 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.
  - Typically, the SQL schema in which the relations are declared is implicitly specified in the environment in which the CREATE TABLE statements are executed. Alternatively, we can explicitly attach the schema name to the relation name, separated by a period. For example,

Data Definition and Data Types

CREATE TABLE COMPANY.EMPLOYEE

Or

CREATE TABLE EMPLOYEE

- The relations declared through CREATE TABLE statements are called **base tables** (or **base relations**); this means that the table and its rows 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 table (relation).

Data Definition and Data Types

```
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) );
```

Figure
SQL CREATE TABLE data definition statements for defining the COMPANY schema

Data Definition and Data Types

```
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) );
```

Data Definition and Data Types

- **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 :**
    - Integer numbers of various sizes (INTEGER or INT, and SMALLINT).
    - 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.

Data Definition and Data Types

- **Character Strings:**
  - Fixed length string, CHAR(n) or CHARACTER(n), where n is the number of characters. A shorter string is padded with blank characters to the right.
  - Varying length string, 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.
  - Strings can be compared lexicographically and concatenate using || operator.
  - Character large object, CHARACTER LARGE OBJECT or CLOB is also variable-length string 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.

Data Definition and Data Types

- **Bit String:**
  - Fixed length BIT(n) or varying length BIT VARYING(n), where n is the maximum number of bits. The default for n is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings, such as, B‘10101’.
  - Variable-length bit string BINARY LARGE OBJECT or BLOB is used to specify 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.
- **Boolean:**
  - 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.

Data Definition and Data Types

- **Date and Time:**
  - 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.
  - Only valid dates and times should be allowed. Dates and times can be compared – 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 ‘2014-09-27’.
  - A data type TIME(i) 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.

Data Definition and Data Types

- **Timestamp:**
  - 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.
  - Literal values are represented by single-quoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time; for example, TIMESTAMP '2014-09-27 09:12:47.648302'.
- **Interval:**
  - 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.
- DATE, TIME, and TIMESTAMP data types can be cast or converted to string formats for comparison.

Data Definition and Data Types

- **CREATE DOMAIN and CREATE TYPE in SQL:**
  - A domain can be declared, and the domain name can be used with the attribute specification. 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) as data type for attributes.
  - There is also a CREATE TYPE command, which can be used to create user defined data types. These can then be used either as data types for attributes, or as the basis for creating tables. It is often used in conjunction with specifying object database features.
  - Creating domains and types may not be available in some implementations of SQL.

Specifying Constraints in SQL

- **Specifying Attribute Constraints and Attribute Defaults:**
  - A constraint **NOT NULL** is specified if NULL is not permitted for a particular attribute. This is always implicitly specified for the attributes that are part of the primary key.
  - It is also possible to define a default value for an attribute 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. If no default clause is specified, the default *default value* is NULL for attributes that do not have the NOT NULL constraint.
  - Another type of constraint can restrict attribute or domain values using the **CHECK** clause following an attribute or domain definition. For example, Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

Specifying Constraints in SQL

- The CHECK clause can also be used in conjunction with the CREATE DOMAIN statement. For example, CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM > 0 AND D_NUM < 21);

- **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 (unique) keys, also known as candidate keys. The UNIQUE clause can also be specified directly for a unique key if it is a single attribute, as in the following example:

    Dname VARCHAR(15) UNIQUE,

Specifying Constraints in SQL

- 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 updated. 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. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE.
- It is the responsibility of the database designer to choose the appropriate action and to specify it in the database schema.

Specifying Constraints in SQL

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

Specifying Constraints in SQL

- **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. It is also possible to temporarily defer a constraint until the end of a transaction.
- **Specifying Constraints on Tuples Using CHECK:**
  - An additional CHECK clauses at the end of a CREATE TABLE statement can also be used. These can be called row-based constraints because they apply to each row individually and are checked whenever a row is inserted or modified. For example, CHECK (Dept_create_date <= Mgr_start_date);

Basic Retrieval Queries in SQL

- The SELECT statement in SQL is used to retrieve data from the database.
- The SELECT statement is not the same as the SELECT operation of relational algebra.
- SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values. Hence, in general, an SQL table is not a set of tuples, because a set does not allow two identical members; rather, it is a **multiset** (sometimes called a **bag**) of tuples.
- Some SQL relations are constrained to be sets because a key constraint has been declared or because the DISTINCT option has been used with the SELECT statement.

Basic Retrieval Queries in SQL

- **The SELECT-FROM-WHERE Structure:**
  - The basic form of the SELECT statement is formed from three clauses SELECT, FROM, and WHERE as given below.

    SELECT <attribute list>

    FROM <table list>

    WHERE <condition>

    <attribute list> is a list of attribute names whose values are to be retrieved by the query.

    <table list>is a list of the relation names required to process the query.

    <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query. Only those tuples that satisfy the condition – that is, those tuples for which the condition evaluates to TRUE – are selected.

Basic Retrieval Queries in SQL

- Logical comparison operators for comparing attribute values with one another and with literal constants are =, <=, >, >=, and <>.
- **Query 1:** Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

  SELECT Bdate, Address

  FROM EMPLOYEE

  WHERE Fname = 'John' **AND** Minit = 'B' **AND** Lname = 'Smith';
- **Query 2:** Retrieve first name and address of all employees who work for the 'Research' department.

  SELECT Fname, Address

  FROM EMPLOYEE, DEPARTMENT

  WHERE Dname = 'Research' AND Dnumber = Dno;

  Here, Dname = 'Research' is a **selection condition** and Dnumber = Dno is called a **join condition**.

Basic Retrieval Queries in SQL

- **Query 3:** 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.

  SELECT Pnumber, Dnum, Lname, Address, Bdate

  FROM PROJECT, DEPARTMENT, EMPLOYEE

  WHERE Dnum = Dnumber **AND** Mgr_ssn = Ssn **AND** Plocation = ‘Stafford’;

Basic Retrieval Queries in SQL

- **Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables:**
  - If the same name is used for two (or more) attributes in different tables, we must qualify the attribute name with the relation name to prevent ambiguity by prefixing the relation name to the attribute name and separating the two by a period.
  - Suppose that, 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, to prevent ambiguity, Query2 above would be rephrased as.

    SELECT Fname, EMPLOYEE.Name, Address

    FROM EMPLOYEE, DEPARTMENT

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

Basic Retrieval Queries in SQL

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

  SELECT EMPLOYEE.Fname, EMPLOYEE.Lname, EMPLOYEE.Address

  FROM EMPLOYEE, DEPARTMENT

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

- We can also **rename** the table names to shorter names by creating an alias for each table name to avoid repeated typing of long table names. For example,

  SELECT E.Fname, E.Lname, E.Address

  FROM EMPLOYEE AS E, DEPARTMENT AS D

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

Basic Retrieval Queries in SQL

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

  The relation names E and S are called **aliases** or **tuple variables** for the EMPLOYEE relation. An alias can also directly follow the relation name, for example, EMPLOYEE E.

  It is also possible to rename relation attributes within the query by giving them aliases. For example, FROM EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno). Here, Fn becomes an alias for Fname, Mi for Minit, Ln for Lname, and so on.

Basic Retrieval Queries in SQL

- **Unspecified WHERE Clause and Use of the Asterisk:**
  - A missing WHERE clause indicates no condition on tuple. Hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result.
  - 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.
  - To select all EMPLOYEE Ssns, we write

    SELECT Ssn

    FROM EMPLOYEE;
  - To select all combinations of EMPLOYEE Ssn and DEPARTMENT Dname, we write

    SELECT Ssn, Dname

    FROM EMPLOYEE, DEPARTMENT;

Basic Retrieval Queries in SQL

- If we specify all the attributes in the SELECT clause without WHERE, we get the actual CROSS PRODUCT (except for duplicate elimination, if any).
- To retrieve all the attribute values of the selected tuples, we just specify an asterisk (*), which stands for all the attributes.
- The * can also be prefixed by the relation name. For example, EMPLOYEE.* refers to all attributes of the EMPLOYEE.
- To retrieve all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5, we write

  SELECT *

  FROM EMPLOYEE

  WHERE Dno = 5;

Basic Retrieval Queries in SQL

- To retrieve 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, and

  SELECT *

  FROM EMPLOYEE, DEPARTMENT

  WHERE Dname = 'Research' **AND** Dno = Dnumber;

- To find CROSS PRODUCT of the EMPLOYEE and DEPARTMENT, we write

  SELECT *

  FROM EMPLOYEE, DEPARTMENT;

Basic Retrieval Queries in SQL

- **Removing Duplicates in SELECT Query:**
  - If we do want to eliminate duplicate tuples in 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.
  - In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not. Specifying SELECT with neither ALL nor DISTINCT is equivalent to SELECT ALL. For example, to retrieve the salary of every employee, we write

    SELECT ALL Salary

    FROM EMPLOYEE;

    And, to retrieve distinct salary values, we write

    SELECT DISTINCT Salary

    FROM EMPLOYEE;

Basic Retrieval Queries in SQL

- **Set Operations in SQL:**
  - SQL has directly incorporated some of the set operations from mathematical set theory. There are set union (UNION), set difference (EXCEPT), and set intersection (INTERSECT) operations.
  - 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 type 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.
  - 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).

Basic Retrieval Queries in SQL

- For example, Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.

```
(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’ );
```

Basic Retrieval Queries in SQL

- **Substring Pattern Matching:**
  - The **LIKE** comparison operator can be used for string pattern matching. We describe patterns by using two special characters: percent (%) matches any substring and underscore (_) matches any character.
  - **Examples:**

    'Hous%' matches any string beginning with "Hous".

    '%Hous' matches any string ending with "Hous".

    '%Hous%' matches any string containing "Hous" as a substring.

    '---' matches any string of exactly three characters.

    '---%' matches any string of at least three characters.

    '%---' matches any string of at most three characters.
  - SQL allows us to search for mismatches instead of matches by using the **NOT LIKE** comparison operator.

Basic Retrieval Queries in SQL

- To retrieve all employees whose address is in Houston, Texas, we write

  SELECT Fname, Lname

  FROM EMPLOYEE

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

- To find all employees who were born during the 1970s, we write

  SELECT Fname, Lname

  FROM EMPLOYEE

  WHERE Bdate **LIKE** '_ _ 7 _ _ _ _ _ _ _';

- If underscore or % is needed in the string, the character should be preceded by an escape character. For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF'. In fact, Any character not used in the string can be chosen as the escape character.

Basic Retrieval Queries in SQL

- 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.

- **Arithmetic Operators:**
  - The standard arithmetic operators {+, −, *, and /} can be applied attributes with numeric domains. For example, to show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10% raise, we write

    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’;

Basic Retrieval Queries in SQL

- 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.

- **BETWEEN Comparison Operators:**
  - To retrieve all employees in department 5 whose salary is between $30,000 and $40,000, we write

    SELECT *

    FROM EMPLOYEE

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

    Here, (Salary BETWEEN 30000 AND 40000) is equivalent to the condition ((Salary >= 30000) AND (Salary <= 40000)).

Basic Retrieval Queries in SQL

- **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.
  - To retrieve name of employees and their dependent name, ordered by first name and, within each first name of the employee, ordered alphabetically by dependent name, we write

    SELECT Fname, Lname, Dependent_name

    FROM EMPLOYEE, DEPENDENT

    WHERE Ssn = Essn

    ORDER BY Fname **ASC**, Dependent_name **DESC**;
  - The default order is ascending order. We can specify the keyword **DESC** if we want to see the result in a descending order. The keyword **ASC** can be used to specify ascending order explicitly.

INSERT, DELETE, and UPDATE

- The three commands used to modify the database are INSERT, DELETE, and UPDATE.
- **The INSERT Command:**
  - In its simplest form, INSERT is used to add a single tuple (row) to a relation (table).
  - 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,

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

INSERT, DELETE, and UPDATE

- A second form of the INSERT statement allows the user to specify explicit attribute names. 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. For example,

  INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn) VALUES (‘Richard’, ‘Marini’, 4, ‘653298653’);
- Attributes not specified are set to their DEFAULT or to NULL.
- 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. For example,

  INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn) VALUES (‘Richard’, ‘Marini’, 4, ‘653298653’), (‘Nawaraj’, ‘Paudel’, 4, ‘653298658’);

INSERT, DELETE, and UPDATE

- A variation of the INSERT command inserts multiple tuples into a relation. For example,

```
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;
```

- Another variation for loading data is to create a new table TNEW that has the same attributes as an existing table T, and load some of the data currently in T into TNEW. For example,

```
CREATE TABLE D5EMPS LIKE EMPLOYEE
(SELECT E.*
FROM EMPLOYEE AS E
WHERE E.Dno = 5) WITH DATA;
```

INSERT, DELETE, and UPDATE

- **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. For example, DELETE FROM EMPLOYEE WHERE Dno = 5;
  - 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.
  - 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. For example, DELETE FROM EMPLOYEE;

INSERT, DELETE, and UPDATE

- **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.
  - 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

    UPDATE PROJECT SET Plocation = ‘Bellaire’, Dnum = 5 WHERE Pnumber = 10;

INSERT, DELETE, and UPDATE

- Several tuples can be modified with a single UPDATE command. An example is to give all employees in the 'Research' department a 10% raise in salary, as shown below

  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.