Database Management System (DBMS)

Avinash V. Gondal

email: avinash.gondal@gmail.com

Module 4 : SQL

Contents :

- Overview of SQL
- Data Definition Commands
- Set operations
- Aggregate function
- Null values
- Data Manipulation commands
- Data Control commands
- Views in SQL
- Nested and complex queries

Introduction

Introduction to SQL

- Main language for relational DBMSs.
- Main characteristics:
  - relatively easy to learn;
  - non-procedural - you specify *what* information you require, rather than *how* to get it;
  - essentially free-format;
  - consists of standard English words like SELECT, INSERT, and UPDATE;
  - can be used by range of users.

History

- Name SQL presently expanded as Structured Query Language.
- Originally called as SEQUEL (Structured English QUEry Language)
- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- ANSI and ISO standard SQL:
  - SQL-86 or ANSI 1986 or SQL1
  - SQL-92 or SQL2
  - SQL-99 →well organized standard
  - Other standards such as SQL3 have been proposed but not fully endorsed by industry.
  - SQL3 is a superset of SQL-92. Therefore, whatever worked in an implementation of SQL-92 should also work in an implementation of SQL3
  - Release of SQL3 fell significantly behind schedule and was only finalized in 1999 . Now known as SQL-99.

Introduction to SQL

- SQL is a comprehensive database language .
- Both as a DDL and DML language.
- SQL statements are divided into following categories :
  - **DDL** (Data Definition Language): define the schema of the database (create, alter and drop).
  - **DML** (Data Manipulation Language): provides commands to manipulate the database (insert, update, delete).
  - **DQL** (data Query Language) **:** enables the users to query one or more tables to get the information they want (select).
  - **DCL** (Data Control Statement) : provides commands to control the user access to the database objects (commit, rollback, grant, revoke).

Introduction to SQL

- Provides facilities for
  - defining views on the database
  - specifying security and authorization
  - defining integrity constraints
  - specifying transaction controls
- SQL can be used interactively or embedded in a high-level language (e.g. C, C++,Java).

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.

Schema and Catalog Concepts in SQL

- # SQL Schema :
- Early versions of SQL→ all tables(relations) were considered part of the same schema.
- The concept of an SQL schema was incorporated starting with SQL2 in order to group together tables and other constructs that belong to the same database application.
- 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 → tables, constraints, views, domains, and other constructs that describe the schema.
- A schema is created via the CREATE SCHEMA statement, which can include all the schema elements definitions.
- Example:

CREATE SCHEMA COMPANY AUTORIZATION Jsmith ;

Not all users are authorized to create schema

- SQL Catalog :
- SQL2 uses a concept of catalog→ a named collection of schema in an SQL environment.
- 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.
- Integrity constraints → only if they exist in schemas within the same catalog.
- Schemas within the same catalog can also share certain elements, such as domain definitions.

Attribute Data Types and Domains in SQL

Attribute Data Types in SQL

- The basic data types available for attributes include :

- Numeric Data Types
- Character-string Data Types
- Bit-string Data Types
- Boolean Data Types
- Date and Time Data Types
- Timestamp Data Type
- Interval Data Type

Numeric Data Types

- Numeric data types include:

- Integer numbers of various sizes :
  - INTEGER (or INT) → integer numerical with precision 10.
  - SMALLINT → integer numerical with precision 5.
  - BIGINT → integer numerical with precision 19.
- Floating Point (real) numbers of various precision :
  - REAL or FLOAT
  - 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.

    j = the scale, is the number of digits after the decimal point
  - The default for scale is zero, and the default for precision is implementation-defined.

Character-string Data Types

- Character-string data types are:

- Fixed length :
  - CHAR (n) or CHARACTER (n)

    where, n = number of characters
- Varying length :
  - VARCHAR (n) or CHAR VARYING (n) or CHARACTER VARYING (n)

    where n = maximum number of characters
- When specifying a literal string value, it is placed between single quotation marks and it is case sensitive.
- For fixed length strings, a shorter string is padded with blank characters to the right.
- Padded blanks are generally ignored when strings are compared.
- For comparison purposes → alphabetic order.
- Concatenation operator denoted by ||.
- Another data type called CHARACTER LARGE OBJECT or CLOB is also available to specify columns that have large text values, such as documents.
  - e.g. NOTES_E_COPY CLOB (200KB)

Bit-string Data Types

- Bit-string data types are:

- Fixed length :
  - BIT (n) where, n = number of bits
- Varying length :
  - BIT VARYING (n)

    where n = 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 quotation marks but preceded by a B to distinguish them from character string.
  - e.g. B '10101'
- For Bit strings whose length is a multiple of 4 can be specified in hexadecimal notation, where literal string is preceded by X and each hexadecimal character represents 4 bits.
- Another data type called BINARY LARGE OBJECT or BLOB is also available to specify columns that have large binary values, such as images.
  - e.g. MY_PHOTO BLOB (10MB)

Boolean Data Types

- Boolean data types has the traditional values of TRUE or FALSE.
- We generally use BIT data type for Boolean values.
- 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.

Date and Time Data Types

- New data types date and time are added in SQL2.

- DATE :
  - The DATE data type has ten positions, and its components are YEAR, MONTH and DAY in the form YYYY-MM-DD.
- TIME :
  - 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 by the SQL implementation.
- The < (less than) comparison can be used with dates or times.
- Literal values are represented by single quoted strings preceded by keyword DATE or TIME.
  - e.g. DATE '2012-09-14' or TIME '09:59:59'
- TIME(i) :
  - Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii….

Date and Time Data Types

- TIME WITH TIME ZONE :
  - The 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 SQL session.

TIMESTAMP Data Types

- TIMESTAMP data type 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.

  - e.g. TIMESTAMP '2012-09-14 09:59:59 648302'

INTERVAL Data Types

- Specifies an interval -- a relative value that can be used to increment or decrement an absolute value of a date, time or timestamp.
- Can be DAY/TIME intervals or YEAR/MONTH intervals
- Can be positive or negative when added to or subtracted from an absolute value, the result is an absolute value

Domains in SQL

- It is possible to specify the data type of each attribute directly as shown in figure B.
- Alternatively a domain can be declared , the domain name 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) in figure B, for 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.

Example : The COMPANY database

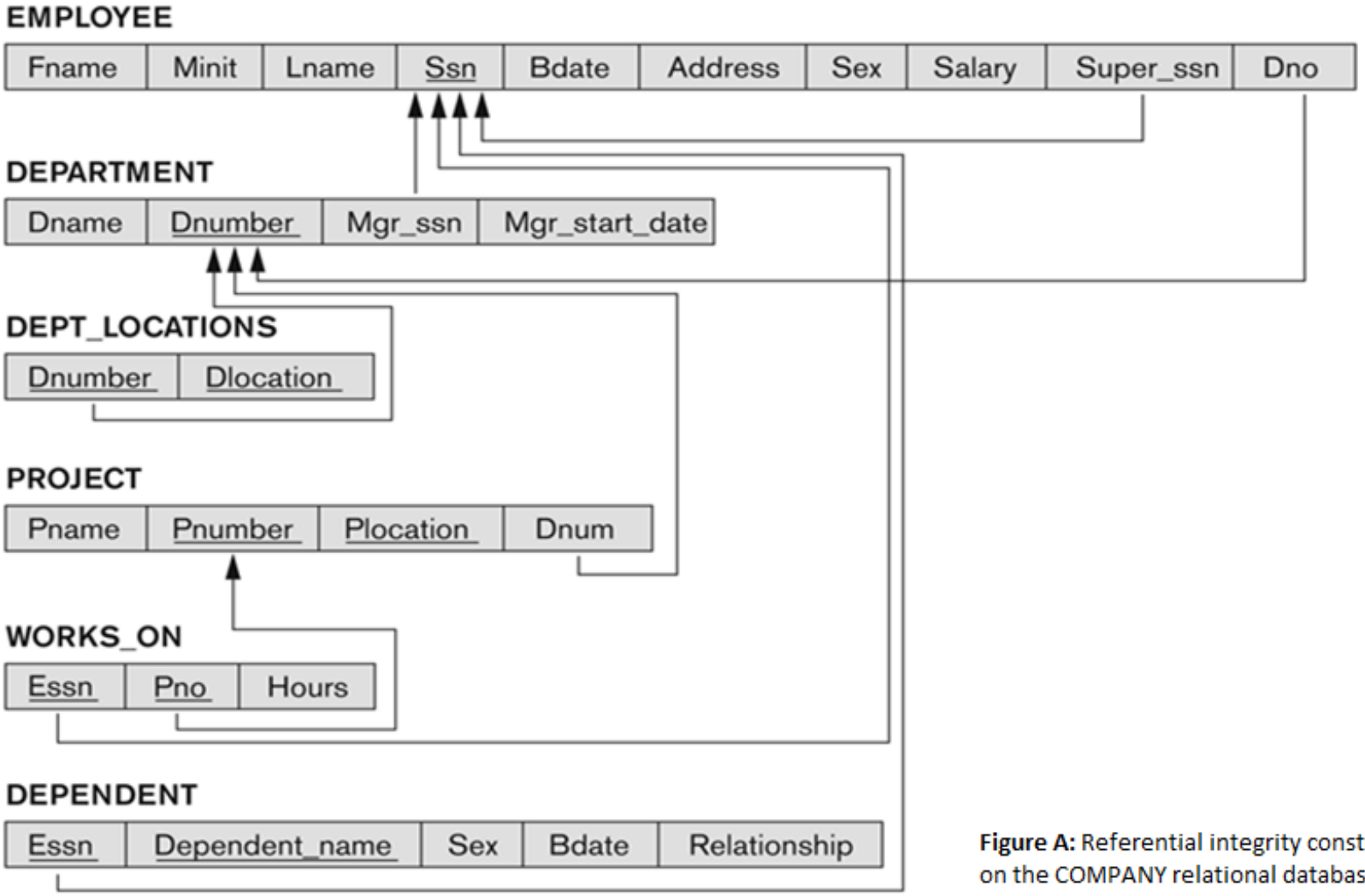

Figure A: Referential integrity constraints displayed on the COMPANY relational database schema.

Example

Figure B:
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure: A

```
CREATE TABLE EMPLOYEE
       ( FNAME          VARCHAR(15)      NOT NULL ,
         MINIT          CHAR ,
         LNAME          VARCHAR(15)      NOT NULL ,
         SSN            CHAR(9)          NOT NULL ,
         BDATE          DATE
         ADDRESS        VARCHAR(30) ,
         SEX            CHAR ,
         SALARY         DECIMAL(10,2) ,
         SUPERSSN       CHAR(9) ,
         DNO            INT              NOT NULL ,
    PRIMARY KEY (SSN) ,
   FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
    FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE DEPARTMENT
       ( DNAME          VARCHAR(15)      NOT NULL ,
         DNUMBER        INT              NOT NULL ,
         MGRSSN         CHAR(9)          NOT NULL ,
         MGRSTARTDATE   DATE ,
       PRIMARY KEY (DNUMBER) ,
       UNIQUE (DNAME) ,
       FOREIGN KEY (MGRSSN) 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) ) ;
```

Example

Figure B: (continued) SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure: A

```
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) ) ;
```

The CREATE TABLE Command in SQL

- The CREATE TABLE command is used to specify a new relation by giving it a name and specifying attributes and initial constraints.
- Attributes→ attribute name→ a data type→ attribute constraint, 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 ALTER TABLE command.
- See figure B.
- 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, by writing……

  CREATE TABLE COMPANY.EMPLOYEE….

  rather than,

  CREATE TABLE EMPLOYEE……

The CREATE TABLE Command in SQL

- We can explicitly (rather than implicitly) make the EMPLOYEE table part of the COMPANY schema.
- The relations declared through CREATE TABLE statements are called base tables (base relations).
- 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 figure B, 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: FK Super_ssn in the EMPLOYEE table→ circular reference.

  FK Dno in the EMPLOYEE refers to the DEPARTMENT table, which is not have been created yet.
- To deal with this type of problem, these constraints can be left out of the initial CREATE TABLE statement, and the added later using the ALTER TABLE statement.

Specifying Constraints in SQL

- This section describes the basic constraints that can be specified in SQL as part of table creation.
- These include key and referential integrity constraints, as well as restrictions on attribute domains and NULLs, and constraints on individual tuples within a relation.

Specifying Attribute Constraints and Attribute Defaults

The DEFAULT Clause in SQL

- Because SQL allows NULLs as attribute values, a *constraint* NOT NULL may be 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* of each relation, but it can be specified for any other attributes whose values are required not to be NULL, as shown in Figure B.
- 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.

The DEFAULT Clause in SQL

- Figure C, 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.

Example

FIGURE C

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

```
CREATE TABLE EMPLOYEE
        ( . . . ,
          DNO                 INT     NOT NULL     DEFAULT 1,
        CONSTRAINT EMPPK
          PRIMARY KEY (SSN) ,
        CONSTRAINT EMPSUPERFK
          FOREIGN KEY (SUPERSSN) 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
        ( . . . ,
          MGRSSN    CHAR(9)  NOT NULL DEFAULT  '888665555' ,
          . . . ,
          CONSTRAINT DEPTPK
            PRIMARY KEY (DNUMBER) ,
          CONSTRAINT DEPTSK
            UNIQUE (DNAME),
          CONSTRAINT DEPTMGRFK
            FOREIGN KEY (MGRSSN) 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 ) ;
```

The CHECK Clause in SQL

- Used to specify user-defined constraints
- Assume that dept. numbers are from 0 to 99.

```
create table DEPARTMENT (
    …
    Dnumber INTEGER      Default 0
        CHECK (Dnumber>=0 AND Dumber<=99),
    …);
```

- CHECK clause can also be used in conjunction with the CREATE DOMAIN STATEMENT

```
CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM>0 AND D_NUM <21);
```

Specifying Key and Referential Integrity Constraints

Key Constraints

- Keys and referential integrity constraints are very important, there are special clauses within the CREATE TABLE statement to specify them.
- Some examples to illustrate the specification of keys and referential integrity are shown in Figure B.
- 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 (instead of the way it is specified in Figure B):

Dnumber INT **PRIMARY KEY;**

Key Constraints

- The **UNIQUE** clause specifies alternate (secondary) keys, as illustrated in theDEPARTMENT and PROJECT table declarations in Figure B.
- 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 Options

- **Causes** of referential integrity violation for a foreign key FK (consider the `Mgr_ssn` of `DEPARTMENT`).
  - **ON DELETE**: when deleting the foreign tuple
    - What to do when deleting the manager tuple in `EMPLOYEE` ?
  - **ON UPDATE:** when updating the foreign tuple
    - What to do when updating/changing the SSN of the manager tuple in `EMPLOYEE` is changed ?
- **Actions** when the above two causes occur.
  - **SET NULL**: the `Mgr_ssn` is set to null.
  - **SET DEFAULT**: the `Mgr_ssn` is set to the default value.
  - **CASCADE**: the `Mgr_ssn` is updated accordingly
    - If the manager is deleted, the department is also deleted.

Giving Names to Constraints

- Figure C also illustrates how 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.

Specifying Constraints on Tuples Using CHECK

- In addition to key and referential integrity constraints, which are specified by special keywords, 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 in Figure B had an additional attribute Dept_create_date, which stores the date when the department was created.

CHECK (Dept_create_date <= Mgr_start_date);

Specifying Constraints on Tuples Using CHECK

- CHECK can also be a clause of the entire table.

```
create table DEPARTMENT (
…
Dept_create_date date,
Mgr_start_date   date,
CHECK (Dept_create_date <= Mgr_start_date)
);
```

- The CHECK clause can also be used to specify more general constraints using the CREATE ASSERTION statement of SQL.

Schema Change Statements in SQL

Schema Change Statements in SQL

- In this section, we give an overview of the **schema evolution commands** available in SQL, which can be used to alter a schema by adding or dropping tables, constraints, and other schema elements.
- The DROP Command
- The ALTER Command

The DROP Command

- The DROP command can be used to drop named schema elements, such as tables, domains or constraints.
- One can also drop a schema.

  DROP SCHEMA
- There are two drop behavior options:
  - CASCADE
  - RESTRICT
- To remove the COMPANY database schema, and all its tables, domains, and other elements, the CASCADE option is used as follows:

  DROP SCHEMA COMPANY CASCADE;
- If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed.

The DROP Command

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

DROP TABLE DEPENDENT CASCADE;

- DROP table command not only deletes all records in the table if successful, but also removes the table definition from the catalog.
- If it is desired to delete the records, but leave the table definition for future use, the DELETE command should be used instead of DROP TABLE.
- The DROP command can also be used to drop other data types of named schema elements, such as constraints or domains.

The ALTER Command

- The definition of a base table or of other named schema elements can be changed by using the ALTER command. For base tables, the possible **alter table actions** include :
  - adding or dropping a column (attribute).
  - changing a column definition.
  - adding or dropping table constraints.
- For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema , we can use the command:

  ALTER TABLE COMPANY.EMPLOYEE **ADD COLUMN** Job VARCHAR(12);

The ALTER Command

- The database users must still enter a value for the new attribute Job for each EMPLOYEE tuple.
  - This can be done either by specifying a default clause or by using the UPDATE command.
  - If no default clause is specified, the new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute.
- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
- For example, the following command removes the attribute Address from the EMPLOYEE base table:

  ALTER TABLE COMPANY.EMPLOYEE **DROP COLUMN** Address **CASCADE**;

The ALTER Command

- It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause. The following examples illustrate this clause:

  ALTER TABLE COMPANY.DEPARTMENT
  ALTER COLUMN Mgr_ssn **DROP DEFAULT**;

ALTER TABLE COMPANY.DEPARTMENT
ALTER COLUMN Mgr_ssn **SET DEFAULT** '333445555';

Basic Retrieval Queries in SQL

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement.
  - This is *not the same as* the SELECT operation of the relational algebra.
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values.
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples.
- SQL relations can be constrained to be sets by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query.

Retrieval Queries in SQL (contd.)

- A **bag** or **multi-set** is like a set, but an element may appear more than once.
  - Example: {A, B, C, A} is a bag. {A, B, C} is also a bag that also is a set.
  - Bags also resemble lists, but the order is irrelevant in a bag.
- Example:
  - {A, B, A} = {B, A, A} as bags.
  - However, [A, B, A] is not equal to [B, A, A] as lists.

Retrieval Queries in SQL (contd.)

- Basic form of the SQL SELECT statement is called a *mapping* or a SELECT-FROM-WHERE *block*

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

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

Retrieval Queries in SQL (contd.)

- In SQL, the basic logical comparison operators are =, <, <=, >, >=, and <>.
- The corresponding relational algebra operators =, <, ≤, >, ≥, and ≠, respectively.
- C/C++ programming language operators =, <, <=, >, >=, and !=.
- The main syntactic difference is the *not equal* operator. SQL has additional comparison operators that we will present gradually.

Relational Database Schema

We will use the example queries specified on the schema of the following figure D:

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Populated Sample Database

We will refer to the sample database state shown in following figure E:

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Simple SQL Queries

Simple SQL Queries

- Basic SQL queries correspond to using the following operations of the relational algebra:
  - SELECT
  - PROJECT
  - JOIN
- All subsequent examples use the COMPANY database

Simple SQL Queries

- Example of a simple query on one relation
- Query 0: Retrieve the birthdate and address of employee whose name is 'John B. Smith'.

```
Q0: SELECT    BDATE, ADDRESS
    FROM      EMPLOYEE
    WHERE     FNAME='John' AND MINIT='B' AND
              LNAME='Smith';
```

  - Similar to a SELECT-PROJECT pair of relational algebra operations:
    - The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition
  - However, the result of the query may contain duplicate tuples.

Result

BDATE	ADDRESS
1965-01-09	731 Fondren, Houston, TX

Simple 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;
```

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

Result

FNAME	LNAME	ADDRESS
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

Simple SQL Queries

- 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 birthdate.

```
Q2:     SELECT    PNUMBER, DNUM, LNAME, BDATE, ADDRESS
        FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE     DNUM=DNUMBER AND MGRSSN=SSN
                  AND PLOCATION='Stafford';
```

- In Q2, there are two join conditions
  - The join condition DNUM=DNUMBER relates a project to its controlling department
  - The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department.

Result :

PNUMBER	DNUM	LNAME	ADDRESS	BDATE
10	4	Wallace	291 Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291 Berry, Bellaire, TX	1941-06-20

Ambiguous Attribute Names, Aliasing and Tuple Variables

Ambiguous Attribute Names

- 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 multi-table 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.
- To illustrate this, 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, query Q1 would be rephrased as shown in Q1A.
- We must prefix the attributes Name and Dnumber in Q1A to specify which ones we are referring to, because the same attribute names are used in both relations:

Ambiguous Attribute Names

Q1A:

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

- Fully qualified attribute names can be used for clarity even if there is no ambiguity in attribute names. Q1 is shown in this manner as is Q1

Q1:

```
SELECT EMPLOYEE.Fname, EMPLOYEE.LName,
EMPLOYEE.Address
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.DName='Research' AND
DEPARTMENT.Dnumber=EMPLOYEE.Dno;
```

ALIASES

- Some queries need to refer to the same relation twice.
  - In this case, *aliases* are given to the relation name.
- Query 8: For each employee, retrieve the employee's first name, 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 E S
     WHERE      E.SUPERSSN=S.SSN;
```

  - In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation.
  - We can think of E and S as two different *copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*.

ALIASES

- Aliasing can also be used in any SQL query for convenience.
- Can also use the AS keyword to specify aliases:

```
Q8: SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM     EMPLOYEE AS E, EMPLOYEE AS S
    WHERE    E.SUPERSSN=S.SSN;
```

- It is also possible to rename the relation attributes within the query in SQL by giving them aliases.

- For Example, we write

EMPLOYEE AS E (Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)

Result of Q8 :

ALIASES

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

Unspecified WHERE Clause and Use of the Asterisk

Unspecified WHERE Clause

- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected.
  - This is equivalent to the condition WHERE TRUE
- Query 9: Retrieve the SSN values for all employees.

```
Q9:   SELECT   SSN
      FROM     EMPLOYEE;
```

Result :

SSN
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

Unspecified WHERE Clause

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

```
Q10:  SELECT   SSN, DNAME
      FROM     EMPLOYEE, DEPARTMENT;
```

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

Unspecified WHERE Clause

- Result of Q10:

SSN
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

DNAME
Research
Research
Research
Research
Research
Research
Research
Research
Administration
Administration
Administration
Administration
Administration
Administration
Administration
Administration
Headquarters
Headquarters
Headquarters
Headquarters
Headquarters
Headquarters
Headquarters
Headquarters

Use of Asterisk (*)

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

  Example:

  Q1C: Retrieve all the attribute values of any EMPLOYEE who works in DEPARTMENT NUMBER 5.

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

Result of Q1C:

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	B	Smith	123456789	1965-09-01	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

Use of Asterisk (*)

Q1D: 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.

```
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;
```

Tables as Sets in SQL

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

Use of DISTINCT

- SQL does not treat a relation as a set; *duplicate tuples can appear*.
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used.
- In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.
- Q11: Retrieve the salary of every employee

```
Q11:    SELECT    SALARY
        FROM      EMPLOYEE;
```

SALARY
30000
40000
25000
43000
38000
25000
25000
55000

- The same query can also be written as follows

```
Q11:    SELECT    ALL SALARY
        FROM      EMPLOYEE;
```

- Result of Q11:

Use of DISTINCT

Q11A : Retrieve all distinct salary values

```
Q11A:    SELECT    DISTINCT SALARY
         FROM      EMPLOYEE;
```

- Result of Q11A :

SALARY
30000
40000
25000
43000
38000
55000

SET OPERATIONS

- SQL has directly incorporated some set operations.
- There is a union operation (**UNION**), and in *some versions* of SQL there are set difference (**MINUS OR EXCEPT**) and intersection (**INTERSECT**) operations.
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result.*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order.
- The next example illustrates the use of UNION.

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.

SET OPERATIONS

Q4:

```
(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' );
```

- 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).
- Basically, each tuple—whether it is a duplicate or not—is considered as a different tuple when applying these operations.

SET OPERATIONS

The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

(a)

R	A
	a1
	a2
	a2
	a3

S	A
	a1
	a2
	a4
	a5

(b)

T	A
	a1
	a1
	a2
	a2
	a2
	a3
	a4
	a5

(c)

T	A
	a2
	a3

(d)

T	A
	a1
	a2

Substring Pattern Matching and Arithmetic Operators

SUBSTRING COMPARISON

- The LIKE comparison operator is used to compare partial strings. This can be used for string pattern matching.
- Two reserved characters are used: '**%**' (or '*****' in some implementations) replaces an arbitrary number of characters, and '**_**' replaces a single arbitrary character.
- Consider the following query

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

Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
Q12:  SELECT    FNAME, LNAME
      FROM      EMPLOYEE
      WHERE     ADDRESS LIKE '%Houston,TX%';
```

SUBSTRING COMPARISON

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

Here, '5' must be the 3rd character of the string (according to our format for date), so the BDATE value is '__5_____', with each underscore as a place holder for a single arbitrary character.

```
Q12A:   SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    BDATE LIKE '__5_____';
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic.

ARITHMETIC OPERATIONS

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

  Query 13: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q13:  SELECT   FNAME, LNAME, 1.1*SALARY AS Increased_sa
      FROM     EMPLOYEE, WORKS_ON, PROJECT
      WHERE    SSN=ESSN AND PNO=PNUMBER  AND
               PNAME='ProductX';
```

- 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.

ARITHMETIC OPERATIONS

- Another comparison operator, which can be used for convenience, is **BETWEEN**, which is illustrated in Query 14.

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)).

Ordering of Query Results

ORDER BY

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

Query 15: Retrieve a list of employees and the projects they are working on, ordered by the employee's department, and within each department ordered alphabetically by employee last name, first name.

```
Q15:  SELECT    DNAME, LNAME, FNAME, PNAME
      FROM      DEPARTMENT, EMPLOYEE,WORKS_ON,
                PROJECT
      WHERE     DNUMBER=DNO AND SSN=ESSN AND
                PNO=PNUMBER
      ORDER BY  DNAME, LNAME,FNAME;
```

- The default order is in ascending order of values.

ORDER BY

- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default.
- If we want descending order on Dname and ascending order on Lname, Fname, the ORDER BY clause of Q15 can be written as

ORDER BY Dname **DESC**, Lname **ASC**, Fname **ASC**

Summary of Basic SQL Retrieval Queries

Summary of Basic SQL Retrieval Queries

- A simple retrieval query in SQL can consist of up to four clauses, but only the first two, SELECT and FROM, are mandatory.
- The clauses are specified in the following order, with the clauses between square brackets [……] being optional :
-

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

- The SELECT-clause lists the attributes or functions to be retrieved.
- The FROM-clause specifies all relations (tables) needed in the simple query .
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause.
- ORDER BY specifies an order for displaying the result of a query.

Specifying Updates in SQL

Specifying Updates in SQL

- There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

The INSERT Command

- In its simplest form, it is used to add one or more tuples to a relation.
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command.
- Example:

U1: INSERT INTO EMPLOYEE
VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
'98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4);

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple.
  - Attributes with NULL values can be left out.
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, Dno and SSN attributes.

U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, Dno, SSN)
VALUES ('Richard', 'Marini', 4, '653298653');

The INSERT Command

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database.
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation.
- Example: Suppose we want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project. A table WORKS_ON_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in 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        Lname, Pname, Hours
       FROM          PROJECT, WORKS_ON, EMPLOYEE
       WHERE         Pnumber=Pno AND Essn=Ssn;
```

The INSERT Command

- NOTE : The 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, the information in WORKS_ON_INFO *may become outdated*. We have to create a view to keep such a table up-to-date.

The DELETE Command

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

The DELETE Command

- Examples:

U4A:
```
DELETE FROM     EMPLOYEE
WHERE           LNAME='Brown';
```

U4B:
```
DELETE FROM     EMPLOYEE
WHERE           SSN='123456789';
```

U4C:
```
DELETE FROM     EMPLOYEE
WHERE           DNO  IN   (SELECT DNUMBER
                           FROM    DEPARTMENT
                           WHERE DNAME='Research');
```

U4D:
```
DELETE FROM EMPLOYEE;
```

The UPDATE Command

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

The UPDATE Command

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:  UPDATE     PROJECT
     SET        PLOCATION = 'Bellaire', DNUM = 5
     WHERE      PNUMBER=10;
```

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

```
U6:     UPDATE     EMPLOYEE
        SET        SALARY = SALARY *1.1
        WHERE      DNO  IN (SELECT    DNUMBER
                            FROM      DEPARTMENT
                            WHERE     DNAME='Research');
```

The UPDATE Command

- 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 Queries

Comparisons Involving NULL and Three-Valued Logic

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL :
  - **Unknown value**
  - **Unavailable or withheld value**
  - **Not applicable attribute**
- Hence, SQL does not distinguish between the different meanings of NULL.
- Each individual NULL value considered to be different from every other NULL value in the various database records.
- When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN (it may be TRUE or it may be FALSE).
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN

3-valued Logic in SQL

- Standard 2-valued logic assumes a condition can evaluate to either TRUE or FALSE.
- With NULLs a condition can evaluate to UNKNOWN, leading to 3-valued logic.
- Example: Consider a condition EMPLOYEE.DNO = 5; this evaluates for individual tuples in EMPLOYEE as follows:
  - TRUE for tuples with DNO=5
  - UNKNOWN for tuples where DNO is NULL
  - FALSE for other tuples in EMPLOYEE

3-valued Logic in SQL (cont.)

- Combining individual conditions using AND, OR, NOT logical connectives must consider UNKNOWN in addition to TRUE and FALSE.
- Table shows the truth tables for 3-valued logic.

Table 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`

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

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

Result :

FNAME	LNAME
James	Borg

Nested Queries, Tuples, and Set / Multiset Comparisons

Nested Queries, Tuples, and Set / Multiset Comparisons

- **Nested queries :**
  - Complete select-from-where blocks within WHERE clause of another query.
  - **Outer query**

- Comparison operator `IN`
  - Compares value *v* with a set (or multiset) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries (cont'd.)

- **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
      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' );
```

- In general, the nested query will return a table (relation), which is a set or multiset of tuples.

Nested Queries (cont'd.)

- SQL allows the use of tuples of values in comparisons by placing them within parentheses.
  - Place them within parentheses
- Query : Select the Essns of all employees who work the same (project, hours) combination on some project that employee 'John Smith' (whose Ssn = '123456789') works on.

```
SELECT   DISTINCT Essn
FROM     WORKS_ON
WHERE    (Pno, Hours) IN ( SELECT   Pno, Hours
                           FROM     WORKS_ON
                           WHERE    Essn='123456789' );
```

Nested Queries (cont'd.)

- 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 <> .
  - The keyword **ALL** can also be combined with each of these operators.
- Query : Select the names of employees whose salary is greater than the salary of all the employees in department 5:

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

Nested Queries (cont'd.)

- In general, we can have several levels of nested queries.
- The reference to an *unqualified attribute* refers to the relation declared in the **innermost nested query**.

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

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

Correlated Nested Queries

Correlated Nested Queries

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated.*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query.*

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

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

Correlated Nested Queries

- In Q16, the nested query has a different result *for each tuple* in the outer query.
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query. For example, Q16 may be written as in Q16A.

```
Q16A:   SELECT    E.FNAME, E.LNAME
        FROM      EMPLOYEE AS E, DEPENDENT AS D
        WHERE     E.SSN=D.ESSN AND E.SEX=D.SEX AND
                  E.FNAME=D.DEPENDENT_NAME;
```

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries.
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently.

The EXISTS and UNIQUE Functions in SQL

The EXISTS Function in SQL

- EXISTS is used to check whether the result of a query is empty (contains no tuples) or not (contains one or more tuples).
  - Applied to a query, but returns a Boolean result (TRUE or FALSE).
  - Can be used in the WHERE-clause as a condition.
  - EXISTS (Q) evaluates to TRUE if the result of Q has one or more tuple; evaluates to FALSE if the result of Q has no tuples.

The EXISTS Function in SQL

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

```
Q16B:   SELECT   E.Fname, E.Lname
        FROM     EMPLOYEE AS E
        WHERE    EXISTS ( SELECT   *
                          FROM     DEPENDENT AS D
                          WHERE    E.Ssn=D.Essn
                                   AND   E.Sex=D.Sex
                                AND E.Fname=D.Dependent_name);
```

The EXISTS Function in SQL

- Query 7: Retrieve the names of employees who are department managers *and* have at least one dependent.

```
Q7:  SELECT   M.FNAME, M.LNAME
     FROM     EMPLOYEE AS M
     WHERE    EXISTS  (SELECT   *
                       FROM     DEPENDENT
                       WHERE    M.SSN=ESSN)
              AND
              EXISTS (SELECT   *
                      FROM     DEPARTMENT
                      WHERE    M.SSN=MGRSSN) ;
```

The NOT EXISTS Function in SQL

- On the other hand, NOT EXISTS (Q) returns TRUE if there are no tuples in the result of nested query Q, and it returns FALSE otherwise.

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 );
```

UNIQUE Function in SQL

- There is another SQL function, UNIQUE(Q), which returns TRUE if there are no duplicate tuples in the result of query Q; otherwise, it returns FALSE.
- This can be used to test whether the result of a nested query is a set or a multiset.

Explicit Sets and Renaming of Attributes in SQL

Explicit (Literal) Sets in SQL

- It is also possible to use an **explicit set of values** in the WHERE clause, rather than a nested query.
- An **explicit (enumerated) set of values** is enclosed in parentheses.
- Query 17: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

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

Renaming of Attributes in SQL

- In SQL, it is possible to rename any attribute that appears in the result of a query by adding the qualifier **AS** followed by the desired new name.
- Hence, the AS construct can be used to alias both attribute and relation names, and it can be used in both the SELECT and FROM clauses.

Query 8A: Retrieve the last name of each employee and his or her supervisor. The new names will appear as column headers in the query result.

```
Q8A:  SELECT  E.Lname AS Employee_name, S.Lname AS
                Supervisor_name
      FROM    EMPLOYEE AS E, EMPLOYEE AS S
      WHERE   E.Super_ssn=S.Ssn;
```

Joining Tables in SQL and Outer Joins

Joined Tables (Relations) in SQL

- Can specify a "joined relation" in the FROM-clause
  - Looks like any other relation but is the result of a join.
  - Allows the user to specify different types of joins (INNER JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc) – see the next slides.
  - Each join type can specify a different query and produce a different result.

Types of join – INNER JOIN

- This is the regular join operation.
- Joined tuples must satisfy all join conditions.
- Example: Query QJ1: Retrieve the employee names with the names of the department they work for.

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM        DEPARTMENT AS D, EMPLOYEE AS E
WHERE   D.DNUMBER=E.DNO ;
```

This can be written using *joined tables* as follows:

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM     (DEPARTMENT AS D JOIN EMPLOYEE AS E ON
   D.DNUMBER=E.DNO) ;
```

Types of join – OUTER JOIN

- In QJ1, an EMPLOYEE record is joined only if it has a matching DEPARTMENT with D.DNUMBER=E.DNO
- Hence, an EMPLOYEE with NULL for E.DNO will not appear in the query result.
- Also, a DEPARTMENT that has no matching EMPLOYEE records (i.e. currently has no employees) does not appear in the query result.
- OUTER JOINs gives the options to include every EMPLOYEE record or every DEPARTMENT record in the query results.
- A record that does not have a matching joined record will be “padded” with an imaginary “NULL record” from the other table (all its attributes will be NULL).

Types of join – LEFT OUTER JOIN

- Example: Query QJ2: Retrieve the employee names with the names of the department they work for; *every department* must appear in the result even if it has no employees.

This can be written using *joined tables* as follows:

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM    (DEPARTMENT AS D LEFT OUTER JOIN
                EMPLOYEE AS E ON D.DNUMBER=E.DNO) ;
```

Note: An earlier left outer join syntax in ORACLE is as follows:

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM        DEPARTMENT AS D, EMPLOYEE AS E
WHERE   D.DNUMBER += E.DNO ;
```

Types of join – RIGHT OUTER JOIN

- Example: Query QJ3: Retrieve the employee names with the names of the department they work for; *every employee* must appear in the result even they are not currently assigned to a department

This can be written using *joined tables* as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM (DEPARTMENT **AS** D **RIGHT OUTER JOIN** EMPLOYEE **AS** E **ON** D.DNUMBER=E.DNO) ;

Note: An earlier right outer join syntax in ORACLE is as follows:

SELECT E.FNAME, E.LNAME, D.DNAME

FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E

WHERE D.DNUMBER =+ E.DNO ;

Types of join – FULL OUTER JOIN

- Example: Query QJ4: Retrieve the employee names with the names of the department they work for; *every employee* and *every department* must appear in the result

This can be written using *joined tables* as follows:

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM    (DEPARTMENT AS D FULL OUTER JOIN
                EMPLOYEE AS E ON D.DNUMBER=E.DNO) ;
```

Note: An earlier full outer join syntax in ORACLE is as follows:

```
SELECT  E.FNAME, E.LNAME, D.DNAME
FROM        DEPARTMENT AS D, EMPLOYEE AS E
WHERE   D.DNUMBER +=+ E.DNO ;
```

Types of join – NATURAL JOIN

- If the join attributes in both tables *have the same name*, the join condition can be left out (it is automatically added by the system)
- NATURAL JOIN is a form of inner join
- Example: QJ5: We rename DNUMBER in DEPARTMENT to DNO to match the join attribute name (DNO) in EMPLOYEE (we also rename other attributes)
- Implicit join condition is E.DNO = D.DNO

```
SELECT  E.FN, E.LN, E. ADR
FROM    (DEPARTMENT AS D (DNM, DNO, MSSN,
            STRDATE)
        NATURAL JOIN
    EMPLOYEE AS E
(FN,MI,LN,S,BD,ADR,SX,SAL,SU,DNO)) ;
```

Types of join – SELF JOIN

- A query that joins a table to itself, for example, employee table can be joined to itself to find out subordinate - supervisor pairs.
- Used when a table has a foreign key relationship to itself (usually parent-child relationship).
- Must create a table alias and structure the query as if you are joining the table to a copy of itself.
- *FROM table1 alias1, ...*
- Use alias, not table name for select and where clauses.

Example :

```
SELECT E.Fname, S.Fname
FROM    EMPLOYEE E JOIN EMPLOYEE S ON
                                E.Super_ssn=S.Ssn ;
```

Types of join – EQUI JOIN

- SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.
- You may also perform EQUI JOIN by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

Syntax

```
SELECT  *
FROM table1
JOIN table2
[ON (join_condition)];
```

Types of join – CROSS JOIN

- The SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table, if no WHERE clause is used along with CROSS JOIN. This kind of result is called as Cartesian Product.
- If, WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.
- An alternative way of achieving the same result is to use column names separated by commas after SELECT and mentioning the table names involved, after a FROM clause.

Syntax

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Aggregate Functions in SQL

Aggregate Functions

- Include **COUNT, SUM, MAX, MIN, and AVG.**
- These can summarize information from multiple tuples into a single tuple.
- 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) AS SUM_SAL,
              MAX(SALARY) AS HIGH_SAL,
              MIN(SALARY) AS LOW_SAL,
              AVG(SALARY) AS MEAN_SAL
      FROM    EMPLOYEE ;
```

Aggregate Functions (cont.)

- Query 20: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q20:  SELECT MAX(E.SALARY), MIN(E.SALARY),
                  AVG(E.SALARY)
      FROM    EMPLOYEE E, DEPARTMENT D
      WHERE   E.DNO=D.DNUMBER AND
                  D.DNAME='Research' ;
```

Can also be written like :

```
SELECT  MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM    (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE  D.DNAME='Research' ;
```

Aggregate Functions (cont.)

- Queries 21 and 22: Retrieve the total number of employees in the company (Q21), and the number of employees in the 'Research' department (Q22). (Note: COUNT(*) counts the number of selected records)

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

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

Aggregate Functions (cont.)

- **Query 23.** Count the number of distinct salary values in the database.

```
Q23:   SELECT   COUNT (DISTINCT Salary)
       FROM     EMPLOYEE;
```

- Query 5: Retrieve the names of all employees who have two or more dependents

```
Q5:    SELECT   Lname, Fname
       FROM     EMPLOYEE
       WHERE    ( SELECT   COUNT (*)
                  FROM     DEPENDENT
                  WHERE    Ssn=Essn ) >= 2;
```

Grouping : The GROUP BY and HAVING Clauses

Grouping (Partitioning Records into Subgroups)

- In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation.
- Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)* – for example, *employees who work in the same department* (have the same DNO).
- The aggregate functions are applied to each subgroup independently.
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause* .

Grouping (cont.)

- 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 ;
```

  - In Q24, the EMPLOYEE tuples are divided into groups-
    - Each group has same value for the grouping attribute DNO.
  - The COUNT and AVG functions are applied to each such group of tuples separately (see Figure next slide).
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples.
  - If NULLs exists in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.

Grouping (cont.)

Figure

Results of GROUP BY Q24

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

Grouping (cont.)

- A join condition can be used with grouping.
- Query 25: For each project, retrieve the project number, 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 ;
```

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

The HAVING-clause

- Sometimes we want to retrieve the values of these aggregate functions for only those *groups that satisfy certain conditions.*
- The **HAVING-**clause is used for specifying a selection condition on groups (rather than on individual tuples).
- Query 26: For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project (Figure – next two slides).

```
Q26:  SELECT PNUMBER, PNAME, COUNT(*)
      FROM     PROJECT, WORKS_ON
      WHERE   PNUMBER=PNO
      GROUP BY  PNUMBER, PNAME
      HAVING COUNT(*) > 2 ;
```

The HAVING-clause

(b)

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the WHERE clause but before applying HAVING

These groups are not selected by the HAVING condition of Q26.

Figure : Results of HAVING (b)Q26

The HAVING-clause

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

The HAVING-Clause (cont.)

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.

```
Q27:  SELECT    Pnumber, Pname, COUNT (*)
      FROM      PROJECT, WORKS_ON, EMPLOYEE
      WHERE     Pnumber=Pno AND Ssn=Essn AND Dno=5
      GROUP BY  Pnumber, Pname;
```

- We must be extra careful when two different conditions apply (one to the function in the SELECT clause and another to the function in the HAVING clause).

The HAVING-Clause (cont.)

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:  SELECT     Dnumber, COUNT (*)
      FROM       DEPARTMENT, EMPLOYEE
      WHERE      Dnumber=Dno AND Salary>40000 AND
                   ( SELECT     Dno
                     FROM       EMPLOYEE
                     GROUP BY   Dno
                     HAVING     COUNT (*) > 5);
```

Summary of SQL Queries

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

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>] ;

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved.
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries, as well as joined tables.
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause.
- GROUP BY specifies grouping attributes.
- HAVING specifies a condition for selection of groups.
- ORDER BY specifies an order for displaying the query result.
  - Conceptually, a query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause and ORDER BY.

Views in SQL

Views in SQL

- A view is a “virtual” table that is *derived* from other tables.
- These other tables can be *base tables* or previously defined views.
- A view does not necessarily exist in physical form; it is considered to be a **virtual table.**
- Allows for limited update operations (since the table may not physically be stored).
- Allows full query operations.
- A convenience for defining complex operations once and reusing the definition.
- Can also be used as a security mechanism.

Specification of Views

SQL command: **`CREATE VIEW`**

- a virtual table (view) name.
- a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations).
- a query to specify the view contents.

SQL Views: An Example

- Specify a virtual DEPT_INFO table to summarize departmental information .
- Makes it easier to query without having to specify the aggregate functions, GROUP BY, and HAVING.

```
CREATE VIEW DEPT_INFO(DEPT_NAME, NO_EMPS,
TOTAL_SAL) AS
SELECT DEPT_NAME, COUNT(*), SUM(SALARY)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER = DNO
GROUP BY DNAME;
```

DEPT_INFO

Dept_name	No_of_emps	Total_sal

Querying the View

- We can specify SQL retrieval queries on a view table, same as on a base table:

```
SELECT DEPT_NAME
FROM DEPT_INFO
WHERE NO_OF_EMPS > 100;
```

- Can also specify joins and other retrieval operations on the view.

SQL Views: Another Example

- Specify a virtual WORKS_ON table (called WORKS_ON_NEW), with EMPLOYEE and PROJECT names (instead of numbers) .
- This makes it easier to query by names without having to specify the two join conditions.

```
CREATE VIEW WORKS_ON_NEW AS
SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER
GROUP BY PNAME;
```

WORKS_ON _NEW

Fname	Lname	Pname	Hours

Querying a View (cont.)

- We can specify SQL retrieval queries on a view table, same as on a base table:

```
V1:     SELECT FNAME, LNAME
        FROM WORKS_ON_NEW
        WHERE PNAME='Research';
```

- When no longer needed, a view can be dropped:

```
DROP WORKS_ON_NEW;
```

View Implementation

- View implementation is hidden from the user.
- Two main techniques

1. Query modification:

- DBMS automatically modifies the view query into a query on the underlying base tables.
- Example : V1 can be written as

```
SELECT Fname, Lname
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn=Essn AND Pno=Pnumber
AND Pname='Research';
```

- Disadvantage:

  Inefficient for views defined via complex queries

  - Especially if many queries are to be applied to the view within a short time period.

View Implementation (cont.)

2. View materialization:

- Involves physically creating and keeping a temporary table that holds the view query result.
- Assumes that other queries on the view will follow.
- Concerns:
  - Maintaining correspondence between the base tables and view when the base tables are updated.
- Strategy:
  - Incremental update of the temporary view table.
- The view is generally kept as a materialized (physically stored) table as long as it is being queried.
- If the view is not queried for a certain period of time, the system may then automatically remove the physical table and recompute it from scratch when future queries reference the view.

Updating of Views

- All views can be queried for retrievals, but many views cannot be updated.
- Update on a view on a single table without aggregate operations:
    - If view includes key and NOT NULL attributes, view update may map to an update on the base table.
- Views involving joins and aggregate functions are generally not updatable unless they can be mapped to unique updates on the base tables.

Checking Views for Updatability

- When a user intends to update a view, must add the clause **`WITH CHECK OPTION`** at the end of the CREATE VIEW statement.
  - This allows the system to check for updatability.
  - If view is not updatable, and error will be generated.
  - If view is updatable, system will create a mapping strategy to process view updates.
- It is also possible to define a view table in the **FROM clause** of an SQL query. This is known as an **in-line view**. In this case, the view is defined within the query itself.

University Questions

- What is view in SQL, how it is defined ? Discuss the problem that may arise when we attempt to update a view. How views are implemented? [5 T - 10M]
- Write a note on following [5M each]
  - 'View' in SQL [2 T]
  - Group by and order by clause in SQL. [2 T]
  - Aggregate functions in SQL [1 T]
- For remaining questions based on SQL queries refer the question bank. [10M each]