CO226: Database Systems

Structured Query Language (SQL)

Sampath Deegalla
dsdeegalla@pdn.ac.lk

31st July 2014

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL (Structured Query Language)

- SQL is a comprehensive database language.
- It supports both
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
- It includes features such as
  - views
  - security
  - authorization
  - transaction control
  - embedding SQL

SQL vs. Relational algebra/calculus

- SQL includes some features from relational algebra and it is greatly based on tuple relational calculus
- SQL is user friendly than both relational algebra and calculus
- In relational algebra, user should specify how to execute the query operations
- In SQL, user specifies what the result is to be

SQL vs. Relational algebra/calculus

- SQL includes some features from relational algebra and it is greatly based on tuple relational calculus
- SQL is user friendly than both relational algebra and calculus
- In relational algebra, user should specify how to execute the query operations
- In SQL, user specifies what the result is to be

SQL vs. Relational algebra/calculus

- SQL includes some features from relational algebra and it is greatly based on tuple relational calculus
- SQL is user friendly than both relational algebra and calculus
- In relational algebra, user should specify how to execute the query operations
- In SQL, user specifies what the result is to be

SQL vs. Relational algebra/calculus

- SQL includes some features from relational algebra and it is greatly based on tuple relational calculus
- SQL is user friendly than both relational algebra and calculus
- In relational algebra, user should specify how to execute the query operations
- In SQL, user specifies what the result is to be

SQL vs. Relational algebra/calculus

- SQL includes some features from relational algebra and it is greatly based on tuple relational calculus
- SQL is user friendly than both relational algebra and calculus
- In relational algebra, user should specify how to execute the query operations
- In SQL, user specifies what the result is to be

Terminology

SQL	Relational Model
table	relation
row	tuple
column	attribute

CREATE SCHEMA

- Specifies a new database schema by giving it a name
  - `CREATE SCHEMA COMPANY;`
  - In MySQL the following statments can be used to create a database schema
    - `CREATE SCHEMA COMPANY;`
    - `CREATE DATABASE COMPANY;`
    - CREATE SCHEMA is a synonym for CREATE DATABASE

CREATE TABLE

- Specifies a base relation (table) by giving it a name, specifying each of its attributes and initial constraints
- Each attribute is given a name and a data type to specify its domain of values
- A constraint `NOT NULL` may be specified on an attribute to disallow null values

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(10) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE );
```

CREATE TABLE

- Specifies a base relation (table) by giving it a name, specifying each of its attributes and initial constraints
- Each attribute is given a name and a data type to specify its domain of values
- A constraint `NOT NULL` may be specified on an attribute to disallow null values

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(10) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE );
```

CREATE TABLE

- Specifies a base relation (table) by giving it a name, specifying each of its attributes and initial constraints
- Each attribute is given a name and a data type to specify its domain of values
- A constraint NOT NULL may be specified on an attribute to disallow null values

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(10) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE );
```

CREATE TABLE

- Specifies a base relation (table) by giving it a name, specifying each of its attributes and initial constraints
- Each attribute is given a name and a data type to specify its domain of values
- A constraint `NOT NULL` may be specified on an attribute to disallow null values

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(10) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE );
```

CREATE TABLE

- Specifies a base relation (table) by giving it a name, specifying each of its attributes and initial constraints
- Each attribute is given a name and a data type to specify its domain of values
- A constraint `NOT NULL` may be specified on an attribute to disallow null values

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(10) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE );
```

CREATE TABLE

- Key attributes can be specified via the PRIMARY KEY
- Use UNIQUE phrases to indicate alternate keys/unique values
- Use FOREIGN KEY to specify referential integrity

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9) NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

CREATE TABLE

- Key attributes can be specified via the `PRIMARY KEY`
- Use `UNIQUE` phrases to indicate alternate keys/unique values
- Use `FOREIGN KEY` to specify referential integrity

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9) NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

CREATE TABLE

- Key attributes can be specified via the `PRIMARY KEY`
- Use `UNIQUE` phrases to indicate alternate keys/unique values
- Use `FOREIGN KEY` to specify referential integrity

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9) NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

CREATE TABLE

- Key attributes can be specified via the `PRIMARY KEY`
- Use `UNIQUE` phrases to indicate alternate keys/unique values
- Use `FOREIGN KEY` to specify referential integrity

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9) NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

CREATE TABLE

- Key attributes can be specified via the PRIMARY KEY
- Use UNIQUE phrases to indicate alternate keys/unique values
- Use FOREIGN KEY to specify referential integrity

```
CREATE TABLE DEPARTMENT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9) NOT NULL,
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

Attribute Data Types in SQL

- The basic data types include numeric, character string, bit string, boolean, date and time.
- Numeric data types

  INTEGER/INT and SMALLINT integer numbers of various sizes

  FLOAT/REAL and DOUBLE PRECISION floating-point (real) numbers of various precision

  DECIMAL(i,j)/DEC(i,j) or NUMERIC(i,j) formatted numbers: i - decimal digit, j-number of digit after decimal point

Attribute Data Types in SQL

- The basic data types include numeric, character string, bit string, boolean, date and time.
- Numeric data types

  INTEGER/INT and SMALLINT integer numbers of various sizes

  FLOAT/REAL and DOUBLE PRECISION floating-point (real) numbers of various precision

  DECIMAL(i,j)/DEC(i,j) or NUMERIC(i,j) formatted numbers: i - decimal digit, j-number of digit after decimal point

Attribute Data Types in SQL

- The basic data types include numeric, character string, bit string, boolean, date and time.
- Numeric data types

  INTEGER/INT and SMALLINT integer numbers of various sizes

  FLOAT/REAL and DOUBLE PRECISION floating-point (real) numbers of various precision

  DECIMAL(i,j)/DEC(i,j) or NUMERIC(i,j) formatted numbers: i - decimal digit, j-number of digit after decimal point

Attribute Data Types in SQL

- The basic data types include numeric, character string, bit string, boolean, date and time.
- Numeric data types

  INTEGER/INT and SMALLINT integer numbers of various sizes

  FLOAT/REAL and DOUBLE PRECISION floating-point (real) numbers of various precision

  DECIMAL(i,j)/DEC(i,j) or NUMERIC(i,j) formatted numbers: i - decimal digit, j-number of digit after decimal point

Attribute Data Types in SQL

- Character-string data types

  CHAR(n)/CHARACTER(n) fixed length character-string: n-number of characters

  VARCHAR(n)/CHAR VARING(n)/CHARACTER VARYING(n) varying length character-string: n-maximum number of characters

  CHARACTER LARGE OBJECT/CLOB large text values, e.g., documents. In mysql TEXT data type is used

Attribute Data Types in SQL

- Character-string data types

  CHAR(n)/CHARACTER(n) fixed length character-string: n-number of characters

  VARCHAR(n)/CHAR VARING(n)/CHARACTER VARYING(n) varying length character-string: n-maximum number of characters

  CHARACTER LARGE OBJECT/CLOB large text values, e.g., documents. In mysql TEXT data type is used

Attribute Data Types in SQL

- Character-string data types

  CHAR(n)/CHARACTER(n) fixed length character-string: n-number of characters

  VARCHAR(n)/CHAR VARING(n)/CHARACTER VARYING(n) varying length character-string: n-maximum number of characters

  CHARACTER LARGE OBJECT/CLOB large text values, e.g., documents. In mysql TEXT data type is used

Attribute Data Types in SQL

- Character-string data types

  CHAR(n)/CHARACTER(n) fixed length character-string: n-number of characters

  VARCHAR(n)/CHAR VARING(n)/CHARACTER VARYING(n) varying length character-string: n-maximum number of characters

  CHARACTER LARGE OBJECT/CLOB large text values, e.g., documents. In mysql TEXT data type is used

Attribute Data Types in SQL

- Bit-string data types

  BIT(n) fixed length bit string
  BIT VARYING(n) varying length bit string
  BINARY LARGE OBJECT/BLOB large binary values, e.g., images

Attribute Data Types in SQL

- Bit-string data types
  - BIT(n) fixed length bit string
  - BIT VARYING(n) varying length bit string
  - BINARY LARGE OBJECT/BLOB large binary values, e.g., images

Attribute Data Types in SQL

- Bit-string data types

  BIT(n) fixed length bit string

  BIT VARYING(n) varying length bit string

  BINARY LARGE OBJECT/BLOB large binary values, e.g., images

Attribute Data Types in SQL

- Bit-string data types

  BIT(n) fixed length bit string
  BIT VARYING(n) varying length bit string
  BINARY LARGE OBJECT/BLOB large binary values, e.g., images

Additional Data Types

- BOOLEAN for TRUE/FALSE values
- DATE

DATE	Made up of year-month-day in the format yyyy-mm-dd
TIME	Made up of hour:minute:second in the format hh:mm:ss
TIME(i)	Made up of hour:minute:second plus i additional digits specifying fractions of a second. format is hh:mm:ss:ii...i
TIMESTAMP	Has both DATE and TIME components

Additional Data Types

- BOOLEAN for TRUE/FALSE values
- DATE

DATE	Made up of year-month-day in the format yyyy-mm-dd
TIME	Made up of hour:minute:second in the format hh:mm:ss
TIME(i)	Made up of hour:minute:second plus i additional digits specifying fractions of a second. format is hh:mm:ss:ii...i
TIMESTAMP	Has both DATE and TIME components

Additional Data Types

- BOOLEAN for TRUE/FALSE values
- DATE

DATE	Made up of year-month-day in the format yyyy-mm-dd
TIME	Made up of hour:minute:second in the format hh:mm:ss
TIME(i)	Made up of hour:minute:second plus i additional digits specifying fractions of a second. format is hh:mm:ss:ii...i
TIMESTAMP	Has both DATE and TIME components

Additional Data Types

- BOOLEAN for TRUE/FALSE values
- DATE

DATE	Made up of year-month-day in the format yyyy-mm-dd
TIME	Made up of hour:minute:second in the format hh:mm:ss
TIME(i)	Made up of hour:minute:second plus i additional digits specifying fractions of a second. format is hh:mm:ss:ii...i
TIMESTAMP	Has both DATE and TIME components

Additional Data Types

- BOOLEAN for TRUE/FALSE values
- DATE

DATE	Made up of year-month-day in the format yyyy-mm-dd
TIME	Made up of hour:minute:second in the format hh:mm:ss
TIME(i)	Made up of hour:minute:second plus i additional digits specifying fractions of a second. format is hh:mm:ss:ii...i
TIMESTAMP	Has both DATE and TIME components

Additional Data Types

- INTERVAL

  INTERVAL Specifies a relative value rather than an absolute value. 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.

- Note: not available in MySQL

Additional Data Types

- INTERVAL

  INTERVAL Specifies a relative value rather than an absolute value. 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.

- Note: not available in MySQL

Specifying constraints in SQL

- `NOT NULL` to indicate null is not permitted for a perticular attribute
- `DEFAULT` to define a default value for an attribute
- `PRIMARY KEY` to specify one or more attributes that make up the primary key of a relation
- `UNIQUE` to specify alternate keys
- `FOREIGN KEY` to specify referential integrity constraints
- `CONSTRAINT` to give name to constraints

REFERENTIAL INTEGRITY OPTIONS

- We can specify RESTRICT, CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)
- This must be specified with either ON DELETE or ON UPDATE
- RESTRICT reject update operation that will cause a violation
- SET NULL set to null value
- CASCADE propaget the update
- SET DEFAULT set to default value

REFERENTIAL INTEGRITY OPTIONS

- Examples

```
CREATE TABLE DEPT (
DNAME VARCHAR(15) NOT NULL,
DNUMBER INTEGER NOT NULL,
MGRSSN CHAR(9),
MGRSTARTDATE DATE,
PRIMARY KEY (DNUMBER),
UNIQUE (DNAME),
FOREIGN KEY (MGRSSN) REFERENCES EMP(SSN)
ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```

REFERENTIAL INTEGRITY OPTIONS

```
CREATE TABLE EMP (
ENAME VARCHAR(30) NOT NULL,
ESSN CHAR(9),
BDATE DATE,
DNO INTEGER DEFAULT 1,
SUPERSSN CHAR(9),
PRIMARY KEY (ESSN),
FOREIGN KEY (DNO) REFERENCES DEPT
ON DELETE SET DEFAULT ON UPDATE CASCADE,
FOREIGN KEY (SUPERSSN) REFERENCES EMP(SSN)
ON DELETE SET NULL ON UPDATE CASCADE
);
```

DROP TABLE

- 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;`

ALTER TABLE

- Used to add an attribute to one of the base relations
- 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
- Example: `ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);`
- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

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 (we will discuss this later)
- 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

- 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 (we will discuss this later)
- 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

- 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 (we will discuss this later)
- 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

- 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 (we will discuss this later)
- 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

- 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 (we will discuss this later)
- 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

- 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 (we will discuss this later)
- 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

- 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

- 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

- 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

- 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

- 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

- 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

Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on one relation

Simple SQL Queries

- Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.
  Q0:
  ```
  SELECT BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'
  ```
- SELECT-clause specifies the projection attributes and the WHERE- clause specifies the selection condition
- However, the result of the query may contain duplicate tuples

Simple SQL Queries

- Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.

  Q0:

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

- SELECT-clause specifies the projection attributes and the WHERE- clause specifies the selection condition
- However, the result of the query may contain duplicate tuples

Simple SQL Queries

- Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.
  Q0:
  ```
  SELECT BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'
  ```
- SELECT-clause specifies the projection attributes and the WHERE- clause specifies the selection condition
- However, the result of the query may contain duplicate tuples

Simple SQL Queries

- Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.
  Q0:
  ```
  SELECT BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'
  ```
- SELECT-clause specifies the projection attributes and the WHERE- clause specifies the selection condition
- However, the result of the query may contain duplicate tuples

Simple SQL Queries

- Query 0: Retrieve the birth date and address of the employee whose name is 'John B. Smith'.
  Q0:
  ```
  SELECT BDATE, ADDRESS
  FROM EMPLOYEE
  WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith'
  ```
- SELECT-clause specifies the projection attributes and the WHERE- clause specifies the selection condition
- However, the result of the query may contain duplicate tuples

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
```

  - (DNAME='Research') is a selection condition
  - (DNUMBER=DNO) is a join condition

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
```

  - (DNAME='Research') is a selection condition
  - (DNUMBER=DNO) is a join condition

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
  ```
  - (DNAME='Research') is a selection condition
  - (DNUMBER=DNO) is a join condition

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
  ```
  - (DNAME='Research') is a selection condition
  - (DNUMBER=DNO) is a join condition

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
  ```
  - (DNAME='Research') is a selection condition
  - (DNUMBER=DNO) is a join condition

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 birth date.
  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

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 birth date.

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

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 birth date.
  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

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 birth date.
  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

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 birth date.
  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

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 birth date.
  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

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in different relations A query that refers to two or more attributes with the same name must qualify the attribute name with the relation name by prefixing the relation name to the attribute name
- Example:
  `EMPLOYEE.NAME, DEPARTMENT.NAME`

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in different relations A query that refers to two or more attributes with the same name must qualify the attribute name with the relation name by prefixing the relation name to the attribute name
- Example:

```
EMPLOYEE.NAME, DEPARTMENT.NAME
```

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in different relations A query that refers to two or more attributes with the same name must qualify the attribute name with the relation name by prefixing the relation name to the attribute name
- Example:
  `EMPLOYEE.NAME, DEPARTMENT.NAME`

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 name, and the name of his or her immediate supervisor.

  Q8:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.

  Q8:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.

  Q8:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.

Q8:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.
  Q8:
  ```
  SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
  FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.
  Q8:
  ```
  SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
  FROM EMPLOYEE AS E, EMPLOYEE AS S (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

- 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 name, and the name of his or her immediate supervisor.
  Q8:
  ```
  SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
  FROM EMPLOYEE AS E, EMPLOYEE AS S (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

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
```

- 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

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
```

- 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

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
```

- 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

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
```

- 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

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
```

- 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

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
```

- 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

UNSPECIFIED WHERE-clause

- 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

- 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

- 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

USE OF *

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

  Examples:

  Q1C:

```
SELECT *
FROM EMPLOYEE
WHERE DNO=5
```

  Q1D:

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER
```

USE OF *

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

Examples:

Q1C:

```
SELECT *
FROM EMPLOYEE
WHERE DNO=5
```

Q1D:

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER
```

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for all the attributes
  Examples:
  Q1C:
  ```
  SELECT *
  FROM EMPLOYEE
  WHERE DNO=5
  ```
  Q1D:
  ```
  SELECT *
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNAME='Research' AND DNO=DNUMBER
  ```

USE OF *

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

  Examples:

  Q1C:

```
SELECT *
FROM EMPLOYEE
WHERE DNO=5
```

  Q1D:

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER
```

USE OF *

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

  Examples:

  Q1C:

```
SELECT *
FROM EMPLOYEE
WHERE DNO=5
```

  Q1D:

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:  SELECT SALARY
FROM EMPLOYEE
Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE
```

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 (EXCEPT/MINUS) 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

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 (EXCEPT/MINUS) 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

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 (EXCEPT/MINUS) 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

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 (EXCEPT/MINUS) 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

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 (EXCEPT/MINUS) 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

SET OPERATIONS

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

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

SET OPERATIONS

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

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

SET OPERATIONS

- Query 4: Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

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

Specifying Updates in SQL

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

INSERT

- 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

INSERT

- Example:

```
U1:  INSERT INTO EMPLOYEE VALUES
('Richard','K','Marini', '653298653', '1962-12-30',
'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, and SSN attributes.

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

INSERT

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database

DELETE

- 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

DELETE

- 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
```

DELETE

- 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
```

DELETE

- 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
```

DELETE

- 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
```

UPDATE

- 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

UPDATE

- 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
```

UPDATE

- 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')
```

- 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

NESTING OF QUERIES

- A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research' )
```

NESTING OF QUERIES

- A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research' )
```

NESTING OF QUERIES

- A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research' )
```

NESTING OF QUERIES

- A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research' )
```

NESTING OF QUERIES

- A complete SELECT query, called a nested query , can be specified within the WHERE-clause of another query, called the outer query
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
FROM DEPARTMENT
WHERE DNAME='Research' )
```

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is not correlated with the outer query

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an unqualified attribute refers to the relation declared in the innermost nested query
- In this example, the nested query is <u>not correlated</u> with the outer query

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 as the employee.

```
Q16:  SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

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 as the employee.

```
Q16:  SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

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 as the employee.

```
Q16:  SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

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 as the employee.

```
Q16:  SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

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 as the employee.

```
Q16:  SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME)
```

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 E, DEPENDENT D
WHERE E.SSN=D.ESSN AND
E.FNAME=D.DEPENDENT_NAME
```

  comparison operator, which is used in conjunction with nested correlated queries difficulty in implementing it efficiently

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 16 in an alternative form that uses EXISTS as Q16B below

THE EXISTS FUNCTION

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

```
Q16B: SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE EXISTS (SELECT *
FROM DEPENDENT
WHERE SSN=ESSN AND FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION

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

```
Q16B: SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE EXISTS (SELECT *
FROM DEPENDENT
WHERE SSN=ESSN AND FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION

- 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)
```

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

THE EXISTS FUNCTION

- 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)
```

- In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected
- EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an explicit (enumerated) set of values in the WHERE-clause rather than a nested query
- 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)
```

EXPLICIT SETS

- It is also possible to use an explicit (enumerated) set of values in the WHERE-clause rather than a nested query
- 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)
```

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 18: Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE SUPERSSN IS NULL
```

- Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 18: Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE SUPERSSN IS NULL
```

- Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Renaming Attributes

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

```
Q8A: SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS
SUPERVISOR_NAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN
```

Renaming Attributes

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

```
Q8A: SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS
SUPERVISOR_NAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN
```

Joined Relations Feature in SQL2

- 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 (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Joined Relations Feature in SQL2

Examples:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNUMBER=DNO) WHERE
DNAME='Research'
```

or as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN
DEPARTMENT AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE DNAME='Research'
```

Joined Relations Feature in SQL2

Examples:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNUMBER=DNO) WHERE
DNAME='Research'
```

or as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN
DEPARTMENT AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE DNAME='Research'
```

Joined Relations Feature in SQL2

Examples:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNUMBER=DNO) WHERE
DNAME='Research'
```

or as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN
DEPARTMENT AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE DNAME='Research'
```

Joined Relations Feature in SQL2

Examples:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO
```

could be written as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNUMBER=DNO) WHERE
DNAME='Research'
```

or as:

```
Q1:  SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN
DEPARTMENT AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE DNAME='Research'
```

Joined Relations

- Query2: For every project located in 'Stafford' list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:  SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
JOIN EMPLOYEE ON MGRSSN=SSN)
WHERE PLOCATION='Stafford'
```

Joined Relations

- Query2: For every project located in 'Stafford' list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:  SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
JOIN EMPLOYEE ON MGRSSN=SSN)
WHERE PLOCATION='Stafford'
```

Joined Relations

- For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor. (Requires all employees be included)

```
SELECT E.LNAME AS EMPLOYEE_LNAME, S.LNAME AS
SUPERVISOR_LNAME
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S ON
E.SUPERSSN=S.SSN)
```

Joined Relations

- For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor. (Requires all employees be included)

```
SELECT E.LNAME AS EMPLOYEE_LNAME, S.LNAME AS
SUPERVISOR_LNAME
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S ON
E.SUPERSSN=S.SSN)
```

AGGREGATE FUNCTIONS

- Include COUNT, SUM, MAX, MIN, and AVG
- Query 19: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q19:  SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE
```

- Some SQL implementations may not allow more than one function in the SELECT-clause

AGGREGATE FUNCTIONS

- Include COUNT, SUM, MAX, MIN, and AVG
- Query 19: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q19:  SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE
```

- Some SQL implementations may not allow more than one function in the SELECT-clause

AGGREGATE FUNCTIONS

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

```
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME='Research'
```

AGGREGATE FUNCTIONS

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

```
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME='Research'
```

AGGREGATE FUNCTIONS

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

```
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
Q: SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME='Research'
```

AGGREGATE FUNCTIONS

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

```
Q21:  SELECT COUNT(*)
FROM EMPLOYEE
Q22:  SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

AGGREGATE FUNCTIONS

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

```
Q21:  SELECT COUNT(*)
FROM EMPLOYEE
Q22:  SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

AGGREGATE FUNCTIONS

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

```
Q21:  SELECT COUNT(*)
FROM EMPLOYEE
Q22:  SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

GROUPING

- 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)
- The function is 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

- 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 having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

GROUPING

- 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 having the same value for the grouping attribute DNO
- The COUNT and AVG functions are applied to each such group of tuples separately
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
- A join condition can be used in conjunction with grouping

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 functions are applied after the joining of the two relations

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 functions are applied after the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these 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)

THE HAVING-CLAUSE

- 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.

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

THE HAVING-CLAUSE

- 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.

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

SUBSTRING COMPARISON

- The LIKE comparison operator is used to compare partial strings
- Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character

SUBSTRING COMPARISON

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

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

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 each works in, 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;
```

ORDER BY

- The default order is in ascending order of values
- 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

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Views (Virtual Tables) in SQL

- A view is a single virtual table that is derived from other tables.
- The other tables could be base tables or previously defined view.
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.
- There are no limitations on querying a view
- The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.
- The view attribute names can be inherited from the attribute names of the tables in the defining query.

Example:

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
      FROM EMPLOYEE, PROJECT, WORKS_ON
      WHERE SSN=ESSN AND PNO=PNUMBER;

V2:   CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
      AS SELECT DNAME, COUNT (*), SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

- In V1, the names of the view attribute names are inherited
- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query.

Example:

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
      FROM EMPLOYEE, PROJECT, WORKS_ON
      WHERE SSN=ESSN AND PNO=PNUMBER;

V2:   CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
      AS SELECT DNAME, COUNT (*), SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

- In V1, the names of the view attribute names are inherited
- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query.

Example:

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
      FROM EMPLOYEE, PROJECT, WORKS_ON
      WHERE SSN=ESSN AND PNO=PNUMBER;

V2:   CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
      AS SELECT DNAME, COUNT (*), SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

- In V1, the names of the view attribute names are inherited
- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query.

Example:

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
      FROM EMPLOYEE, PROJECT, WORKS_ON
      WHERE SSN=ESSN AND PNO=PNUMBER;

V2:   CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
      AS SELECT DNAME, COUNT (*), SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

- In V1, the names of the view attribute names are inherited
- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query.

Example:

```
V1:   CREATE VIEW WORKS_ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
      FROM EMPLOYEE, PROJECT, WORKS_ON
      WHERE SSN=ESSN AND PNO=PNUMBER;

V2:   CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
      AS SELECT DNAME, COUNT (*), SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME;
```

- In V1, the names of the view attribute names are inherited
- In V2, the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

QUERIES ON VIEWS

- Example: Retrieve the last name and first name of all employees who work on 'ProjectX'

```
QV1: SELECT PNAME, FNAME, LNAME
     FROM  WORKS_ON1
     WHERE  PNAME='ProjectX';
```

- Without the view WORKS_ON1, this query specification would require two join conditions
- A view can be defined to simplify frequently occurring queries
- The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified
- Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

- A view is removed using the DROP_VIEW command Example:

```
V1A: DROP VIEW WORKS_ON1;
V2A: DROP VIEW DEPT_INFO;
```

- Views can also be used as a security and authorization mechanism

- A view is removed using the DROP_VIEW command Example:

```
V1A: DROP VIEW WORKS_ON1;
V2A: DROP VIEW DEPT_INFO;
```

- Views can also be used as a security and authorization mechanism

- A view is removed using the DROP_VIEW command Example:

```
VIA: DROP VIEW WORKS_ON1;
V2A: DROP VIEW DEPT_INFO;
```

- Views can also be used as a security and authorization mechanism

- A view is removed using the DROP_VIEW command Example:

```
VIA: DROP VIEW WORKS_ON1;
V2A: DROP VIEW DEPT_INFO;
```

- Views can also be used as a security and authorization mechanism

UPDATING OF VIEWS

- A view update operation may be mapped in multiple ways to update operations on the defining base relations
- The topic of updating views is still an active research area
- Example: Suppose we issue the command in UV1 to update the WORKS_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UV1:  UPDATE WORKS_ON1
      SET  PNAME = 'ProductY'
      WHERE  LNAME = 'Smith' AND FNAME='John' AND
             PNAME ='ProductX';
```

UPDATING OF VIEWS

- A view update operation may be mapped in multiple ways to update operations on the defining base relations
- The topic of updating views is still an active research area
- Example: Suppose we issue the command in UV1 to update the WORKS_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UV1:  UPDATE WORKS_ON1
      SET  PNAME = 'ProductY'
      WHERE  LNAME = 'Smith' AND FNAME='John' AND
             PNAME ='ProductX';
```

UPDATING OF VIEWS

- A view update operation may be mapped in multiple ways to update operations on the defining base relations
- The topic of updating views is still an active research area
- Example: Suppose we issue the command in UV1 to update the WORKS_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UV1:  UPDATE WORKS_ON1
      SET  PNAME = 'ProductY'
      WHERE  LNAME = 'Smith' AND FNAME='John' AND
             PNAME ='ProductX';
```

UPDATING OF VIEWS

- A view update operation may be mapped in multiple ways to update operations on the defining base relations
- The topic of updating views is still an active research area
- Example: Suppose we issue the command in UV1 to update the WORKS_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UV1:  UPDATE WORKS_ON1
      SET  PNAME = 'ProductY'
      WHERE  LNAME = 'Smith' AND FNAME='John' AND
             PNAME ='ProductX';
```

UPDATING OF VIEWS

- A view update operation may be mapped in multiple ways to update operations on the defining base relations
- The topic of updating views is still an active research area
- Example: Suppose we issue the command in UV1 to update the WORKS_ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UV1:  UPDATE WORKS_ON1
      SET  PNAME = 'ProductY'
      WHERE  LNAME = 'Smith' AND FNAME='John' AND
             PNAME ='ProductX';
```

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

- This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:
  - Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'
    - It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted this way

```
UPDATE PROJECT
SET PNAME='ProductY'
WHERE PNAME='ProductX';
```

  - Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple
    - This is most likely the update the user means

```
UPDATE WORKS_ON1
SET PNO=(SELECT PNUMBER FROM PROJECT
         WHERE   PNAME='ProductY')
WHERE  ESSN=(SELECT SSN FROM EMPLOYEE
             WHERE  LNAME='Smith' AND FNAME='John')
       AND
       PNO=  (SELECT PNUMBER FROM PROJECT
              WHERE PNAME='ProductX')
```

- Some view updates may not make much sense; for example, modifying the TOTAL_SAL attribute of DEPT_INFO as in UV2

```
UV2: UPDATE DEPT_INFO
     SET TOTAL_SAL=100000
     WHERE DEPT_NAME='Research';
```

```
UPDATE WORKS_ON1
SET PNO=(SELECT PNUMBER FROM PROJECT
         WHERE   PNAME='ProductY')
WHERE  ESSN=(SELECT SSN FROM EMPLOYEE
             WHERE  LNAME='Smith' AND FNAME='John')
       AND
       PNO=  (SELECT PNUMBER FROM PROJECT
              WHERE PNAME='ProductX')
```

- Some view updates may not make much sense; for example, modifying the TOTAL_SAL attribute of DEPT_INFO as in UV2

```
UV2: UPDATE DEPT_INFO
     SET TOTAL_SAL=100000
     WHERE DEPT_NAME='Research';
```

```
UPDATE WORKS_ON1
SET PNO=(SELECT PNUMBER FROM PROJECT
         WHERE   PNAME='ProductY')
WHERE  ESSN=(SELECT SSN FROM EMPLOYEE
             WHERE  LNAME='Smith' AND FNAME='John')
       AND
       PNO=  (SELECT PNUMBER FROM PROJECT
              WHERE PNAME='ProductX')
```

- Some view updates may not make much sense; for example, modifying the TOTAL_SAL attribute of DEPT_INFO as in UV2

```
UV2: UPDATE DEPT_INFO
     SET TOTAL_SAL=100000
     WHERE DEPT_NAME='Research';
```

```
UPDATE WORKS_ON1
SET PNO=(SELECT PNUMBER FROM PROJECT
         WHERE   PNAME='ProductY')
WHERE  ESSN=(SELECT SSN FROM EMPLOYEE
             WHERE  LNAME='Smith' AND FNAME='John')
       AND
       PNO=  (SELECT PNUMBER FROM PROJECT
              WHERE PNAME='ProductX')
```

- Some view updates may not make much sense; for example, modifying the TOTAL_SAL attribute of DEPT_INFO as in UV2

```
UV2: UPDATE DEPT_INFO
     SET TOTAL_SAL=100000
     WHERE DEPT_NAME='Research';
```

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

- In general, we cannot guarantee that any view can be updated
- A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.
- If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.
- We can make the following general observations:
  - A view with a single defining table is updatable if the view attributes contain the primary key.
  - Views defined on multiple tables using joins are generally not updatable.
  - Views defined aggregate functions are not updatable.

Implementation of Views

- Query Modification
  - Modifying the view query into a query on the underlying base tables.
- View Materialization
  - Creating a temporary view table when the view is first queried.
  - Incrementally updating the view table when base table are updated.

Implementation of Views

- Query Modification
  - Modifying the view query into a query on the underlying base tables.
- View Materialization
  - Creating a temporary view table when the view is first queried.
  - Incrementally updating the view table when base table are updated.

Implementation of Views

- Query Modification
  - Modifying the view query into a query on the underlying base tables.
- View Materialization
  - Creating a temporary view table when the view is first queried.
  - Incrementally updating the view table when base table are updated.

Schema Change Statements in SQL

- The DROP command
  - The DROP command can be used to drop a schema and named schema elements, such as tables, domains, or constraints.
  - Examples
    - `DROP SCHEMA COMANY CASCADE;`
    - `DROP TABLE DEPENDENT CASCADE;`
  - Examples in MySQL
    - `DROP DATABASE | SCHEMA COMANY;` – No support for CASCADE or RESTRICT options
    - `DROP TABLE DEPENDENT CASCADE;`

Schema Change Statements in SQL

- The DROP command
  - The DROP command can be used to drop a schema and named schema elements, such as tables, domains, or constraints.
  - Examples
    - `DROP SCHEMA COMANY CASCADE;`
    - `DROP TABLE DEPENDENT CASCADE;`
  - Examples in MySQL
    - `DROP DATABASE | SCHEMA COMANY;` – No support for CASCADE or RESTRICT options
    - `DROP TABLE DEPENDENT CASCADE;`

Schema Change Statements in SQL

- The ALTER command
  - The ALTER command can be used to change a base table or named schema elements.
  - Examples
    - `ALTER TABLE EMPLOYEE ADD COLUMN Job VARCHAR(12);`
    - `ALTER TABLE EMPLOYEE DROP COLUMN Address CASCADE;`
  - Examples in MySQL
    - `ALTER TABLE EMPLOYEE ADD COLUMN Job VARCHAR(12);`
    - `ALTER TABLE EMPLOYEE DROP COLUMN Address;`

Schema Change Statements in SQL

- The ALTER command
  - The ALTER command can be used to change a base table or named schema elements.
  - Examples
    - `ALTER TABLE EMPLOYEE ADD COLUMN Job VARCHAR(12);`
    - `ALTER TABLE EMPLOYEE DROP COLUMN Address CASCADE;`
  - Examples in MySQL
    - `ALTER TABLE EMPLOYEE ADD COLUMN Job VARCHAR(12);`
    - `ALTER TABLE EMPLOYEE DROP COLUMN Address;`

SQL Triggers

- Objective: to monitor a database and take action when a condition occurs
- Triggers are expressed in a syntax similar to assertions and include the following:
  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)

SQL Triggers: An Example

- A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.SALARY>(SELECT SALARY
                  FROM EMPLOYEE
                  WHERE SSN=NEW.SUPERVISOR_SSN))
INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN,NEW.SSN);
```

SQL Triggers: An Example in MySQL

- Enforce referential integrity using triggers.
- For example we will consider the company database. In employee table dno is a foreign key to the the department table on dnumber. Assume that we need to define foreign as follows

```
create table employee(
--type definitions
foreign key(dno) references department(dnumber) on delete ca
);
```

SQL Triggers: An Example in MySQL

- To perform the above task, we need to write a trigger on the event 'after delete on department', and the action performs on the employee table to make the data consistant.

```
delimiter |

create trigger delete_dno
after delete on department
for each row
delete from employee where dno=old.dnumber;
|

delimiter ;
```

Database Programming

- Objective: to access a database from an application program (as opposed to interactive interfaces)
- Why? An interactive interface is convenient but not sufficient; a majority of database operations are made thru application programs (nowadays through web applications)

Database Programming Approaches

- Embedded commands: database commands are embedded in a general- purpose programming language
- Library of database functions: available to the host language for database calls; known as an API
- A brand new, full-fledged language (minimizes impedance mismatch) e.g. Oracle PL/SQL

Impedance Mismatch

- Incompatibilities between a host programming language and the database model, e.g.,
  - type mismatch and incompatibilities; requires a new binding for each language
  - set vs. record-at-a-time processing ..need special iterators to loop over query results and manipulate individual values

Steps in Database Programming

1. Client program opens a connection to the database server
2. Client program submits queries to and/or updates the database
3. When database access is no longer needed, client program terminates the connection

Database Programming in Php

- commonly used functions are:
  - `mysqli_connect (‘‘hostname’’, ‘‘username’’, ‘‘password’’,‘‘database’’)`: Connects to the MySQL server.
  - `mysqli_query(‘‘connection / link identifier’’, ‘‘query’’)`: Used to send any type of MySQL command to the server.
  - `mysqli_fetch_rows(‘‘results variable from query’’)`: Used to return a row of the entire results of a database query.
  - `mysqli_fetch_array(‘‘results variable from query’’)`: Used to return several rows of the entire results of a database query.
  - `mysqli_error()`: Shows the error message that has been returned directly from the MySQL server.
  - `mysqli_close(‘‘connection / link identifier’’)`: Close the database connection

Example

```
<?php

// connect to MySQL
$con = mysqli_connect("localhost", "username", "password",

// SQL query
$query = "SELECT FName, LName FROM Employee";

$results = mysqli_query($con,$query) or die(mysqli_error()

while ($row = mysqli_fetch_array($results)) {
  echo $row['FName'] . " " . $row['LName'];
  echo "<br>";
}

mysqli_close($con);

?>
```

Database Stored Procedures

- Persistent procedures/functions (modules) are stored locally and executed by the database server (as opposed to execution by clients)
- Advantages:
  - if the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
  - execution by the server reduces communication costs
  - When security is an important issue (e.g. Bank), use stored procedures and functions for all common operations. Applications and users have no access to the database tables directly
  - enhance the modeling power of views

Stored Procedure Constructs

- A stored procedure

```
CREATE PROCEDURE procedure-name (params)
local-declarations
procedure-body;
```

- A stored function

```
CREATE FUNCTION fun-name (params) RETRUNS return-type
local-declarations
function-body;
```

- Calling a procedure or function

```
CALL procedure-name/fun-name (arguments);
```

SP Example in MySQL

- Write a procedure to calculate the averaged employee salary for the company.

```
delimiter |
create procedure avgsalary()
begin
  select avg(salary) as avg_sal
  from employee;
end |
delimiter ;

call avgsalary();
```

SP Example in MySQL

- Write a function to calculate the averaged employee salary for a given department.

```
delimiter |
create function depavgsalary(no int) returns decimal(8,2)
begin
  declare avg decimal(8,2);
  select avg(salary)
  from employee
  where dno=no
  into avg;
  return avg;
end |
delimiter ;

select depavgsalary(5);
```