Basic SQL

Introduction

- We deal with a large amount of data. Need a way to define the database, tables, etc. Process data more effectively. Turn the data into information.

- And this is where **SQL** comes to play!
  - The most commonly used language to interface with a DB system is the **Structured Query Language (SQL)**

- MySQL is a **database management system**
  - Allows you to manage relational databases
  - Open source! by Oracle

What is MySQL?

- We deal with a large amount of data. Need a way to define the database, tables, etc. Process data more effectively. Turn the data into information.
- And this is where **SQL** comes to play!
- MySQL is a **database management system**
  - Allows you to manage relational databases
  - Open source! by Oracle

Install MySQL

- Download MySQL installer from: http://dev.mysql.com/downloads/installer/

- A tutorial shows you step by step how to install MySQL using MySQL Installer: http://www.mysqltutorial.org/install-mysql/

MySQL Workbench

- A unified visual tool for database architects, developers, and DBAs

- https://www.mysql.com/products/workbench/

Introduction

- You have learned how to formulate queries in a mathematical language (relational algebra), it is time to put it into **practice**

- SQL is based on relational algebra

Introduction (cont.)

- A "case-in-point" Example: Find FName, LName of **employees** in the "Research" **department**
  - Relational algebra:

$$\pi_{fname,\ lname}(Employee \bowtie_{dno\ =\ dnumber} \sigma_{dname\ =\ 'Research'}(Department))$$

$$\pi_{fname,\ lname}(\sigma_{dno\ =\ dnumber\ \mathbf{AND}\ dname\ =\ 'Research'}(Employee \times Department))$$

  - SQL:

```
SELECT fname, lname
FROM   employee, department
WHERE  dno=dnumber
  AND  dname='Research'
```

- I don't know about you, but I can clearly see the correlation between Relational Algebra and SQL:
  1. The **FROM** clause specifies a **Cross product**
  2. The **WHERE** clause specifies a selection operation
  3. The **SELECT** clause specifies a projection operation

Introduction (cont.)

- Relational algebra are **low-level** language for DBMS users
  - A sequence of operations
  - User must specify **how**—that is, *in what order* to execute

- This chapter presents practical relational model
  - Based on SQL standard
  - The **SQL** language provides a **high-level** *declarative* language interface
  - User only specifies **what** the result is to be
  - More user-friendly

Introduction (cont.)

- SQL is not only a **query language** - i.e., a language used to retrieve information from a database
- SQL also contain:
  - a **Data Definition Language** - i.e.:
    - a language that is used to define the database and its objects, e.g., tables, views, etc.
  - a **Data Manipulation Language** - i.e.:
    - a language that is used to update and query data
  - a **Data Control Language**
    - a language that is used to grant the permissions to a user to access a certain data in the database

Order of Discussion of Chp 6 and Chp 7

- Data Definition
- Retrieval Query (SELECT)
- Data update (INSERT, DELETE, UPDATE)
- View definition

Today's Lecture

1. SQL for creating schemas and tables
2. An overview of basic data types in SQL
3. How basic constraints are specified
4. Specify retrieval queries (SELECT)
5. SQL commands for insertion, deletion, and update

Data Definition Language Features in SQL

- SQL allow users to perform the following **data definition functions**:
  - **Create** a **database**
  - **Define** **new relations** in a database
  - **Define** **constraints** on attributes in the relations
  - **Alter** the **structure** of (existing) relations

    (This operation can be very complex when there is already data in the relations - it is best done when relations are still empty)

  - **Delete (drop)** **relations**

Creating a Database: the CREATE SCHEMA command

- The **CREATE SCHEMA** command is used to *define* a database schema
  - The description of a database is called the **database schema**
    - Used to group together database tables (= relations)
    - Also contains other constructs (such as constraints)

Figure 2.1
Schema diagram for the database in Figure 1.2.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Creating a Database: the CREATE SCHEMA command (cont.)

- Syntax of the CREAT SCHEMA command in the SQL standard:

CREATE SCHEMA schema_name **AUTHORIZATION** db_user;

To indicate who own the schema

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

- When a database schema is created, **ALL access authorization** is granted to the user db_user (i.e., he is the owner of the database)

Creating a Database: the CREATE SCHEMA command (in MySQL)

- **MySQL** version of **Create Schema**

```
create database database_name ;
```

- The **database** is created by the **root user**

- The authorization is granted separately

- See example:

The CREATE TABLE Command in SQL

- The **CREATE TABLE** command is used to *define* **tables** (= **relations**) within a database schema
- The **relation** created by the **CREAT TABLE** command is initially empty
- You can then use an **INSERT** command to insert data into the relation

```
CREATE TABLE EMPLOYEE
    ( Fname             VARCHAR(15)        NOT NULL,
      Minit             CHAR,
      Lname             VARCHAR(15)        NOT NULL,
      Ssn               CHAR(9)            NOT NULL,
      Bdate             DATE,
      Address           VARCHAR(30),
      Sex               CHAR,
      Salary            DECIMAL(10,2),
      Super_ssn         CHAR(9),
      Dno               INT                NOT NULL,
    PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
    ( Dname             VARCHAR(15)        NOT NULL,
      Dnumber           INT                NOT NULL,
      Mgr_ssn           CHAR(9)            NOT NULL,
      Mgr_start_date    DATE,
    PRIMARY KEY (Dnumber),
    UNIQUE (Dname),
    FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

Data Type in SQL

- Each attribute has a **data type**
- Numeric types:
  - **INTEGER** or **INT**
  - **DECIMAL(*i*, *j*)**
    - DEC(8,3) = fixed point numbers of the format: xxxxx.yyy
- Character string
  - **CHARACTER(n)** or **CHAR(n)**
    - Fixed length character strings (number of characters in string is (always) *n*)
  - **VARCHAR(n)**
    - Variable length character strings (maximum number of characters in string is n)
  - Ex.: To denote string, use single quotes: 'John'
- Bit string
  - **BIT(n)**
    - Fixed length bit string (number of bits in string is (always) n)
  - **BIT VARYING(n)**
    - Variable length bit string (maximum number of bits in string is n)
  - **To denote bit string 10101, use: B'10101'**

Data Type in SQL (cont.)

- Boolean
  - **TRUE**, **FALSE** or **UNKNOWN**
    - if either x or y is NULL , then some logical comparisons evaluate to an UNKNOWN
- Date
  - DATE values are specified as: **DATE'YYYY-MM-DD‘**
  - Must be preceded by the keyword DATE
- Time
  - TIME values are specified as: **TIME'HH:MM:SS'**
  - Must be preceded by the keyword TIME

Specifying Constraints on Relations

- The **CREATE TABLE** command can be used to *specify*:
  - Constraints
  - Default values
    - Specify default values for attributes

- Types of constraints
  - Primary Key constraint
  - Entity Integrity
  - Referential integrity constraint
  - Domain constraint
  - Not-NULL constraint

Specifying Primary Key Constraints

- To define that a given set of **attribute** constitute the **primary key** of the relation, we use:
  - **PRIMARY KEY**(attribute-list), or
  - Dnumber INT **PRIMARY KEY** (if a primary key has a single attribute)

```
CREATE TABLE EMPLOYEE
    ( Fname        VARCHAR(15)      NOT NULL,
      Minit        CHAR,
      Lname        VARCHAR(15)      NOT NULL,
      Ssn          CHAR(9)          NOT NULL,
      Bdate        DATE,
      Address      VARCHAR(30),
      Sex          CHAR,
      Salary       DECIMAL(10,2),
      Super_ssn    CHAR(9),
      Dno          INT              NOT NULL,
    PRIMARY KEY (Ssn),
```

Specifying Primary Key Constraints (cont.)

- Example: multiple attributes in key

```
CREATE TABLE WORKS_ON
        ( Essn          CHAR(9)          NOT NULL,
          Pno           INT              NOT NULL,
          Hours         DECIMAL(3,1)     NOT NULL,
        PRIMARY KEY (Essn, Pno),
```

- The third will fail:

```
insert into test2 values ('111223333', 44, 4.5);
insert into test2 values ('111223333', 23, 3.5);

insert into test2 values ('111223333', 44, 6.5);
```

Specifying Primary Key Constraints (cont.)

- There may be more than one keys in a relation
- The UNIQUE constraints can be used to specify **candidate keys**:
  - **UNIQUE**(attribute-list)

```
CREATE TABLE DEPARTMENT
        ( Dname             VARCHAR(15)        NOT NULL,
          Dnumber           INT                NOT NULL,
          Mgr_ssn           CHAR(9)            NOT NULL,
          Mgr_start_date    DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
```

- Now the second insert will fail:
  - insert into test3 values (4, 'Research')
  - insert into test3 values (5, 'Research')

Specifying Referential Integrity Constraints

- Recall that:

> A **Foreign Key** is a **set of attributes** used to reference (identify) **tuples in *another* relation**
>
> The **referential integrity constraint states** that the **referenced tuples must exist**

- To define that a given set of attribute constitute a foreign key, we use:
  - **FOREIGN KEY** (attribute-list) **REFERENCES** relation(attribute-list)

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

Another Example

```
CREATE TABLE test4
(
   ssn CHAR(9),                /* <-------- key    */
   salary dec(9,2),

   CONSTRAINT Test4PrimKey PRIMARY KEY(ssn)
);

CREATE TABLE test5
(
   essn CHAR(9),
   pno  INTEGER,


   CONSTRAINT Test5ForeignKey
      FOREIGN KEY (essn)
      REFERENCES test4(ssn)  /* Used as foreign key */
);
```

- Now this insert will fail:
  - Insert into test5 values ('111223333', 5);
  - Because there is no tuple in test4 with SSN='111223333'
- To make this insert legal, we must **first** insert the employee:
  - Insert into test4 values ('111223333', 5000.00);

Specifying Referential Integrity Constraints (cont.)

- The attributes used as a foreign key must be specified as:
  - Primary key, or as
  - Unique
- Example: this will cause an **error**:

```
CREATE TABLE test4
(
   ssn CHAR(9),          // Not primary key nor UNIQUE
   salary dec(9,2)
);

CREATE TABLE test5
(
   essn CHAR(9),
   pno  INTEGER,

   CONSTRAINT Test5ForeignKey
      FOREIGN KEY (essn)
      REFERENCES test4(ssn)
);
```

Given Names to Constraints

- A **constraint name** is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint,
- Example:
  - **CONSTRAINT** [constraint-name] **PRIMARY KEY**(attribute-list)
  - **CONSTRAINT** [constraint-name] **UNIQUE**(attribute-list)
  - **CONSTRAINT** [constraint-name]

    FOREIGN KEY (attribute-list)

    REFRENCES relation(attribute-list)

Giving names to constraints is **optional**

Specifying Constraints on Attributes

- SQL also allow the user to define a number of constraints on attribute values (domain constrains)
  - Not Null
    - Specifies that the attribute value cannot have the **NULL** value
  - Default value
  - Domain range
- Example:

```
CREATE TABLE test6
(
   ssn CHAR(9) NOT NULL,
   fname CHAR(30),
   lname CHAR(30)
);
```

- This insert will succeed:
  - insert into test6 values ('123456789', 'John', 'Smith');
- But this insert will fail:
  - insert into test6 values (null, 'John', 'Smith');

Specifying Constraints on Attributes (cont.)

- Specifies a **default value** for an attribute

```
CREATE TABLE test7
(
   ssn CHAR(9) NOT NULL,
   salary DECIMAL(6,2) DEFAULT 5000
);
```

- This is a insert using a **partial set of attributes**:
  - insert into test7(ssn) values ('111223333');
  - The tuple does not contain the **salary attribute**

- The default value 5000 is that assigned to the salary attribute , so effectively:

  insert into test7 values ('111223333', 5000);

Specifying Constraints on Tuples Using CHECK

- Row-based Constraints
  - Apply to each row individually and are checked whenever a row is inserted or modified
- Example

```
CREATE TABLE test8
(
   ssn CHAR(9) NOT NULL,
   dno INTEGER CHECK (dno > 0 and dno < 21)
);
```

- This insert will succeed:
  - insert into test8 values ('111223333', 11);
- This insert will fail:
  - insert into test8 values ('333333333', 44);

SQL: SELECT

- The SQL SELECT Command
  - Is used to retrieve the set of **tuples** that satisfy a given **condition**
  - Is based on **Relational Algebra** and you will see the **join/section/projection** operations reflected in the SELECT command
- The basic form of the SELECT command
  - SELECT-FROM-WHERE block:

```
SELECT attribute_list
FROM   relation_list
WHERE  boolean_expression
```

1. **attribute_list**: a list of attributes from the relations in the relation_list
2. **relation_list**: a list of relations. The **CROSS PRODUCT** of these relations is formed
3. **boolean_expression**: condition

SQL: SELECT (cont.)

```
SELECT attr1, attr2, ...., attrM
FROM   R1, R2, R3, ..., RN
WHERE  boolean_expression
```

- Meaning of the SQL SELECT command:
  - First, form the **Cross product** of the relations R1 × R2 × R3 × ... × RN
  - Next, select all the **tuples** that satisfy the **boolean_expression**
  - Finally, project the **attributes** attr1, attr2, ..., attrM from the **qualifying tuples**

Running Examples

- Example 1: List SSN, Lname and DNO of all **employees**

```
select ssn, lname, dno
from   employee
```

```
SSN       LNAME             DNO
--------- ---------- ----------
123456789 Smith               5
333445555 Wong                5
999887777 Zelaya              4
987654321 Wallace             4
666884444 Narayan             5
453453453 English             5
987987987 Jabbar              4
```

- Exam

```
select dnumber, dname
from   department
```

```
   DNUMBER DNAME
---------- ---------------
         5 Research
         4 Administration
         1 Headquarters
```

Running Examples

- Example 3: See what happens when
"Department" are specified in the r

Observation:

- The **output** is the **cartesian product** of the two relations "employee" and "department"

Query:

```sql
select ssn, lname, dno, dnumber, dname
from employee, department
```

Output:

SSN	LNAME	DNO	DNUMBER	DNAME
123456789	Smith	5	5	Research
333445555	Wong	5	5	Research
999887777	Zelaya	4	5	Research
987654321	Wallace	4	5	Research
666884444	Narayan	5	5	Research
453453453	English	5	5	Research
987987987	Jabbar	4	5	Research
888665555	Borg	1	5	Research
123456789	Smith	5	4	Administration
333445555	Wong	5	4	Administration
999887777	Zelaya	4	4	Administration
987654321	Wallace	4	4	Administration
666884444	Narayan	5	4	Administration
453453453	English	5	4	Administration
987987987	Jabbar	4	4	Administration
888665555	Borg	1	4	Administration
123456789	Smith	5	1	Headquarters
333445555	Wong	5	1	Headquarters
999887777	Zelaya	4	1	Headquarters
987654321	Wallace	4	1	Headquarters
666884444	Narayan	5	1	Headquarters
453453453	English	5	1	Headquarters
987987987	Jabbar	4	1	Headquarters
888665555	Borg	1	1	Headquarters

Running Examples

- Example 4: demonstrates how to perform a **join** operation on **"employee"** and **"department"**

Query:

```
select ssn, lname, dno, dnumber, dname
from employee, department
where dno = dnumber
```

Output:

```
SSN       LNAME             DNO    DNUMBER DNAME
--------- ---------- ---------- ---------- ---------------
888665555 Borg                1          1 Headquarters
999887777 Zelaya              4          4 Administration
987654321 Wallace             4          4 Administration
987987987 Jabbar              4          4 Administration
123456789 Smith               5          5 Research
453453453 English             5          5 Research
666884444 Narayan             5          5 Research
333445555 Wong                5          5 Research
```

Running Examples

- Example 4: demonstrates how to perform a **join** operation on **"employee"** and **"department“**
  - For each tuple (row):

```
dno == dnumber   (as according to the where condition)
```

  - In fact, this is what we have learned before in **Relational Algebra**:

$$R_1 \bowtie_{condition} R_1 = \sigma_{condition}(Department)$$

    - A **join** operation is a **Cross product** followed by a **selection** operation
    - The **FROM** clause in the SQL command specifies the **Cross product** operation
    - The **WHERE** clause in the SQL command specifies the **condition of the σ** operation