CSC 261/461 – Database Systems
Lecture 2

Fall 2017
1. Database System Concepts and Architecture

2. SQL introduction & schema definitions
   - ACTIVITY: Table creation

3. Basic single-table queries
   - ACTIVITY: Single-table queries!

4. Multi-table queries
   - ACTIVITY: Multi-table queries!
Table Schemas

• The **schema** of a table is the table name, its attributes, and their types:

  Product(Pname: string, Price: float, Category: string, Manufacturer: string)

• A **key** is an attribute whose values are unique; we underline a key

  Product(Pname: string, Price: float, Category: string, Manufacturer: string)
• **Database State:**
  – Refers to the *content* of a database at a moment in time.

• **Initial Database State:**
  – Refers to the database state when it is initially loaded into the system.

• **Valid State:**
  – A state that satisfies the structure and constraints of the database.
Database Schema vs. Database State (continued)

• Distinction
  – The *database schema* changes very infrequently.
  – The changes every time the database is updated.
  – *database state*

• Schema is also called intension.
• State is also called extension.
Example of a Database Schema

<table>
<thead>
<tr>
<th>STUDENT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Student_number</td>
</tr>
<tr>
<td>Class</td>
<td>Major</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Course_name</td>
<td>Course_number</td>
</tr>
<tr>
<td>Credit_hours</td>
<td>Department</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PREREQUISITE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Course_number</td>
<td>Prerequisite_number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Section_identifier</td>
<td>Course_number</td>
</tr>
<tr>
<td></td>
<td>Semester</td>
</tr>
<tr>
<td></td>
<td>Year</td>
</tr>
<tr>
<td></td>
<td>Instructor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GRADE_REPORT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Student_number</td>
<td>Section_identifier</td>
</tr>
<tr>
<td></td>
<td>Grade</td>
</tr>
</tbody>
</table>

Figure 2.1
Schema diagram for the database in Figure 1.2.
Example of a database state

**COURSE**

<table>
<thead>
<tr>
<th>Course_name</th>
<th>Course_number</th>
<th>Credit_hours</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro to Computer Science</td>
<td>CS1310</td>
<td>4</td>
<td>CS</td>
</tr>
<tr>
<td>Data Structures</td>
<td>CS3320</td>
<td>4</td>
<td>CS</td>
</tr>
<tr>
<td>Discrete Mathematics</td>
<td>MATH2410</td>
<td>3</td>
<td>MATH</td>
</tr>
<tr>
<td>Database</td>
<td>CS3380</td>
<td>3</td>
<td>CS</td>
</tr>
</tbody>
</table>

**SECTION**

<table>
<thead>
<tr>
<th>Section_identifier</th>
<th>Course_number</th>
<th>Semester</th>
<th>Year</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>MATH2410</td>
<td>Fall</td>
<td>04</td>
<td>King</td>
</tr>
<tr>
<td>92</td>
<td>CS1310</td>
<td>Fall</td>
<td>04</td>
<td>Anderson</td>
</tr>
<tr>
<td>102</td>
<td>CS3320</td>
<td>Spring</td>
<td>05</td>
<td>Knuth</td>
</tr>
<tr>
<td>112</td>
<td>MATH2410</td>
<td>Fall</td>
<td>05</td>
<td>Chang</td>
</tr>
<tr>
<td>119</td>
<td>CS1310</td>
<td>Fall</td>
<td>05</td>
<td>Anderson</td>
</tr>
<tr>
<td>135</td>
<td>CS3380</td>
<td>Fall</td>
<td>05</td>
<td>Stone</td>
</tr>
</tbody>
</table>

**GRADE_REPORT**

<table>
<thead>
<tr>
<th>Student_number</th>
<th>Section_identifier</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>112</td>
<td>B</td>
</tr>
<tr>
<td>17</td>
<td>119</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>85</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>102</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>135</td>
<td>A</td>
</tr>
</tbody>
</table>

**PREREQUISITE**

<table>
<thead>
<tr>
<th>Course_number</th>
<th>Prerequisite_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS3380</td>
<td>CS3320</td>
</tr>
<tr>
<td>CS3380</td>
<td>MATH2410</td>
</tr>
<tr>
<td>CS3320</td>
<td>CS1310</td>
</tr>
</tbody>
</table>

*Figure 1.2* A database that stores student and course information.
Three-Schema Architecture

• Proposed to support DBMS characteristics of:
  – Program-data independence.
  – Support of **multiple views** of the data.

• Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization
Three-Schema Architecture

• Defines DBMS schemas at three levels:
  – **Internal schema** at the internal level to describe physical storage structures and access paths (e.g. indexes).
    • Typically uses a **physical** data model.
  – **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
    • Uses a **conceptual** or an **implementation** data model.
  – **External schemas** at the external level to describe the various user views.
    • Usually uses the same data model as the conceptual schema.
The three-schema architecture

Figure 2.2
The three-schema architecture.

External Level
- External/Conceptual Mapping

Conceptual Level
- Conceptual/Internal Mapping

Internal Level

End Users

External View

... 

External View

Conceptual Schema

Internal Schema

Stored Database
Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user’s external view (e.g. formatting the results of an SQL query for display in a Web page)
Data Independence

• **Logical Data Independence:**
  – The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.

• **Physical Data Independence:**
  – The capacity to change the internal schema without having to change the conceptual schema.
  – For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance.
Data Independence (continued)

• When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

• The higher-level schemas themselves are **unchanged**.
  – Hence, the application programs need not be changed since they refer to the external schemas.
DBMS Languages

• Data Definition Language (DDL)

• Data Manipulation Language (DML)
  – High-Level or Non-procedural Languages: These include the relational language SQL
    • May be used in a standalone way or may be embedded in a programming language
  – Low Level or Procedural Languages:
    • These must be embedded in a programming language
• **Data Definition Language (DDL):**
  – Used by the DBA and database designers to specify the conceptual schema of a database.
  – In many DBMSs, the DDL is also used to define internal and external schemas (views).
• **Data Manipulation Language (DML):**
  – Used to specify database retrievals and updates
  – DML commands (data sublanguage) can be *embedded* in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
    • A library of functions can also be provided to access the DBMS from a programming language
  – Alternatively, stand-alone DML commands can be applied directly (called a *query language*).
Types of DML

• High Level or Non-procedural Language:
  – For example, the SQL relational language
  – Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
  – Also called declarative languages.

• Low Level or Procedural Language:
  – Retrieve data one record-at-a-time;
  – Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.
DBMS Interfaces

• Stand-alone query language interfaces
  – Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)

• Programmer interfaces for embedding DML in programming languages

• User-friendly interfaces
  – Menu-based, forms-based, graphics-based, etc.

• Mobile Interfaces
  – interfaces allowing users to perform transactions using mobile apps
User-Friendly DBMS Interfaces

– Menu-based (Web-based)
  • popular for browsing on the web

– Forms-based
  • designed for naïve users used to filling in entries on a form

– Graphics-based
  • Point and Click, Drag and Drop, etc.
  • Specifying a query on a schema diagram

– Natural language
  • requests in written English

– Combinations of the above:
  • For example, both menus and forms used extensively in Web database interfaces
Other DBMS Interfaces

- Natural language: free text as a query
- Speech: Input query and Output response
- Web Browser with keyword search
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
  - Creating user accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access paths
Typical DBMS Component Modules

Figure 2.3
Component modules of a DBMS and their interactions.

CSC 261, Fall 2017, UR
1. SQL INTRODUCTION & DEFINITIONS
What you will learn about in this section

1. What is SQL?
2. Basic schema definitions
3. Keys & constraints intro
4. ACTIVITY: CREATE TABLE statements
Basic SQL
SQL Introduction

• SQL is a standard language for querying and manipulating data
• SQL stands for Structured Query Language
• SQL is a very high-level programming language
  – This works because it is optimized well!
Basic SQL

• **SQL language**
  – Considered one of the major reasons for the commercial success of relational databases

• **SQL**
  – The origin of SQL is relational predicate calculus called tuple calculus which was proposed initially as the language SQUARE.
  – SQL Actually comes from the word “**SEQUEL (Structured English Query Language)**”
    • Original term used in the paper: “SEQUEL TO SQUARE” by Chamberlin and Boyce. IBM could not copyright that term, so they abbreviated to SQL and copyrighted the term SQL.
  – Now popularly known as “Structured Query language”.

CSC 261, Fall 2017, UR
SQL Data Definition, Data Types, Standards

- Terminology:
  - Table, row, and column used for relational model terms relation, tuple, and attribute

- CREATE statement
  - Main SQL command for data definition
SQL is a...

• **Data Definition Language (DDL)**
  – Define relational *schemata*
  – Create/alter/delete tables and their attributes

• **Data Manipulation Language (DML)**
  – Insert/delete/modify tuples in tables
  – Query one or more tables – discussed next!
A relation or table is a multiset of tuples having the attributes specified by the schema.

Let’s break this definition down.

### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
A multiset is an unordered list (or: a set with multiple duplicate instances allowed)

List: [1, 1, 2, 3]
Set: {1, 2, 3}
Multiset: {1, 1, 2, 3}

i.e. no next(), etc. methods!
### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation.

**NB:** Attributes must have an **atomic** type in standard SQL, *i.e.*, not a list, set, etc.
### Tables in SQL

#### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

A **tuple** or **row** is a single entry in the table having the attributes specified by the schema.

*Also referred to sometimes as a **record***
### Tables in SQL

#### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

The number of tuples is the **cardinality** of the relation.

The number of attributes is the **arity** of the relation.

---

CSC 261, Fall 2017, UR
Data Types in SQL

• Atomic types:
  – Characters: CHAR(20), VARCHAR(50)
  – Numbers: INT, BIGINT, SMALLINT, FLOAT
  – Others: MONEY, DATETIME, …

• Every attribute must have an atomic type
  – Hence tables are flat
A **key** is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation.

- A key is an implicit constraint on which tuples can be in the relation
  - i.e. if two tuples agree on the values of the key, then they must be the same tuple!

```students
Students(sid:string, name:string, gpa:float)
```
NULL and NOT NULL

• To say “don’t know the value” we use **NULL**
  – NULL has (sometimes painful) semantics, more detail later

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bob</td>
<td>3.9</td>
</tr>
<tr>
<td>143</td>
<td>Jim</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Say, Jim just enrolled in his first class.

In SQL, we may constrain a column to be NOT NULL, e.g., “name” in this table
General Constraints

• We can actually specify arbitrary assertions
  – E.g. “There cannot be 25 people in the DB class”

• In practice, we don’t specify many such constraints. Why?
  – Performance!

Whenever we do something ugly (or avoid doing something convenient) it’s for the sake of performance
Summary of Schema Information

• Schema and Constraints are how databases understand the semantics (meaning) of data

• They are also useful for optimization

• SQL supports general constraints:
  – Keys and foreign keys are most important
  – We’ll give you a chance to write the others
ACTIVITY
Acknowledgement

• Some of the slides in this presentation are taken from the slides provided by the authors.
• Many of these slides are taken from cs145 course offered by Stanford University.
• Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.