

CSC 261/461 – Database Systems

Lecture 2

Spring 2017
MW 3:25 pm – 4:40 pm
January 18 – May 3
Dewey 1101

Agenda

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 Schema vs. Database State

- 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 *database state* changes every time the database is updated.
- Schema is also called **intension**.
- State is also called **extension**.

Example of a Database Schema

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
----------------	--------------------	-------

Figure 2.1

Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

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

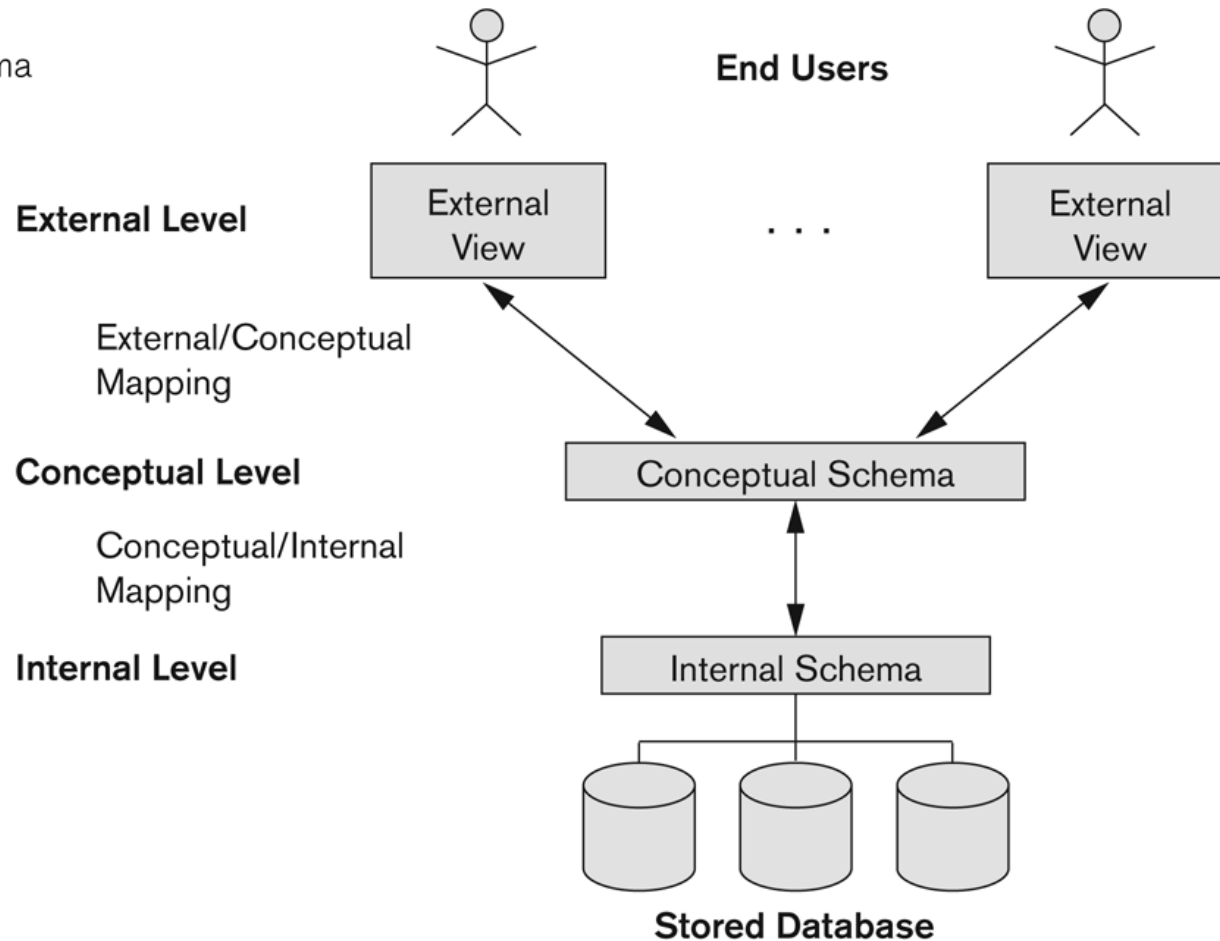
Three-Schema Architecture

- 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.



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

DBMS Languages

- **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).

DBMS Languages

- **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

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
 - **Embedded Approach:** e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)
 - **Procedure Call Approach:** e.g. JDBC for Java, ODBC (Open Database Connectivity) for other programming languages as API's (application programming interfaces)
 - **Database Programming Language Approach:** e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components
 - **Scripting Languages:** PHP (client-side scripting) and Python (server-side scripting) are used to write database programs.

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

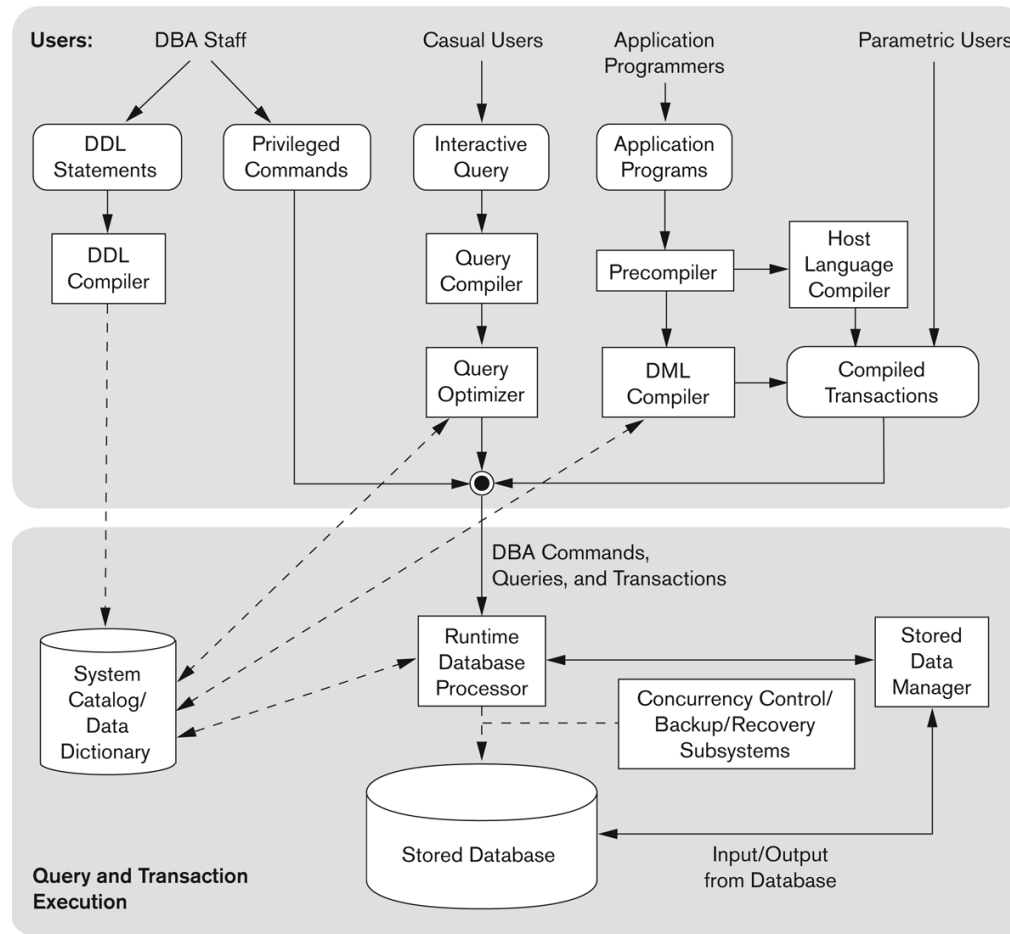


Figure 2.3

Component modules of a DBMS and their interactions.

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”.

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!

Tables in SQL

Product

PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

A relation or table is a multiset of tuples having the attributes specified by the schema

Let's break this definition down

Tables in SQL

Product

PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

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!

Tables in SQL

Product

PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

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

PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

*Also referred to sometimes as a **record***

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

Tables in SQL

Product

PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks
SingleTouch	\$149.99	Canon
MultiTouch	\$203.99	Hitachi

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

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

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

Key constraints

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(sid:string, name:string, gpa:  
float)
```

1. Which would you select as a key?
2. Can we have more than one key?

NULL and NOT NULL

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

```
Students(sid:string, name:string, gpa:  
float)
```

sid	name	gpa
123	Bob	3.9
143	Jim	NULL

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.