CS 6400 A

Database Systems Concepts and Design

Lecture 2

08/20/25

Agenda

1. SQL introduction & schema definitions

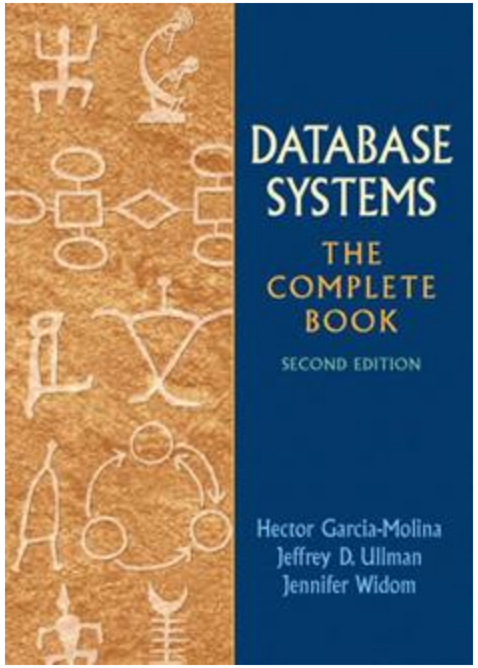
2. Basic single-table queries
   - ACTIVITY: Single-table queries

3. Multi-table queries
   - ACTIVITY: Multi-table queries

Reading Materials

Database Systems: The Complete Book (2nd edition)

- Chapter 2: The Relation Model of Data (2.2- 2.3)
- Chapter 6: The Database Language SQL (6.1-6.2)



Acknowledgement: The following slides have been adapted from CS145 (Intro to Big Data Systems) taught by Peter Bailis.

1. SQL Introduction & Definitions

SQL

- SQL is a standard language for querying and manipulating data

- SQL is a **very high-level** programming language
  - This works because it is optimized well!

- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
  - Vendors support various subsets

SQL stands for
Structured Query Language

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

Tables in SQL

Product

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

A <u>**relation**</u> or <u>**table**</u> 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 <u>**multiset**</u> 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

*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 <u>**cardinality**</u> of the relation

The number of attributes is the <u>**arity**</u> of the relation

Tables in SQL

Q: How many ways are there to represent this relation?

Product

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

- A relation is a set of tuples (not a list)
- A schema is a set of attributes (not a list)
- Hence, the order of tuples or attributes of a relation is immaterial

Data Types in SQL

If CHAR(n) string has fewer than n characters, padded with spaces

- Atomic types:
  - Characters: `CHAR(20), VARCHAR(50)`
  - Numbers: `INT, BIGINT, SMALLINT, FLOAT`
  - Others: `DATE, TIME`, …

- Every attribute must have an atomic type
  - Hence tables are flat

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(<u>Pname</u>: *string*, Price: *float*, Category: *string*, <u>Manufacturer</u>: *string*)

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. Is a key always guaranteed to exist?
3. 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 90 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 (more details later)

Creating a Table in SQL

- To create a table, use CREATE TABLE

```
CREATE TABLE Movies (
  title       CHAR(100),
  year        INT,
  length      INT,
  genre       CHAR(10),
  studioName  CHAR(30),
  producer    INT
);
```

```
CREATE TABLE MovieStar (
  name       CHAR(30),
  address    VARCHAR(30),
  gender     CHAR(1),
  birthdate  DATE
);
```

Modifying relation schemas

- To modify a table, use ALTER TABLE and DROP TABLE

```
DROP TABLE R;

ALTER TABLE MovieStar ADD phone CHAR(16);

ALTER TABLE MovieStar DROP birthdate;
```

Existing tuples will have NULL values for attribute phone

Declaring keys

- Declare one attribute to be a key
- Add separate declaration which attributes form a key
  - Need to use this method for multiple-attribute keys

```
CREATE TABLE MovieStar (
  name        CHAR(30) PRIMARY KEY,
  address  VARCHAR(30),
  gender   CHAR(1),
  birthdate        DATE
);
```

```
CREATE TABLE MovieStar (
  name        CHAR(30),
  address  VARCHAR(30),
  gender   CHAR(1),
  birthdate        DATE,
  PRIMARY KEY (name, address)
);
```

Inserting tuples

A new tuple can be inserted into the relation *R* using an insertion statement.

- For any missing attributes of *R*, the tuple has default values
- If we provide values for all attributes, the list of attributes can be omitted

```
INSERT INTO Movies(title, year, length, genre, studio)
        VALUES ('Ponyo', 2008, 103, 'anime', 'Ghibli');
```

producer will have a NULL default value

Movies(title: *string*, year: *int*, length: *int*, genre: *string, studio: string, producer: int*)

Deleting tuples

- Use a delete statement to delete every tuple satisfying a condition
  - The tuple must be described by a WHERE clause
  - Be careful: omitting the WHERE clause removes all tuples from table

```
DELETE FROM Movies
WHERE year >= 2008
        AND length > 100
        AND genre = 'anime';
```

Updating tuples

- Change the components of existing tuples in the database
    - Multiple assignments are separated by commas

```
UPDATE Movies
SET length = 110, Producer = 123
WHERE title = 'Ponyo'
        AND year = 2008;
```

2. Basic SQL

Simple SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a <u>SFW</u> query.

Simple SQL Query

- Simplest form: ask for tuples in a relation that satisfy a condition

```
Movies(title, year, length, genre, studioName)
```

```
SELECT *
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Simple SQL Query: Projection

- We can replace the * of the SELECT clause with attributes of the relation

```
SELECT title, length
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

title	year	length	genre	studioName
Ponyo	2008	103	anime	Ghibli

title	length
Ponyo	103

<u>Projection</u> is the operation of producing an output table with tuples that have a subset of their prior attributes

Simple SQL Query: Projection

- Use the keyword AS and alias to change an attribute's name

```
SELECT title AS name, length
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

title	year	length	genre	studioName
Ponyo	2008	103	anime	Ghibli

name	length
Ponyo	103

Projection is the operation of producing an output table with tuples that have a subset of their prior attributes

Simple SQL Query: Projection

- Use an expression in place of an attribute

```
SELECT title, length/60 AS lengthHrs
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

title	year	length	genre	studioName
Ponyo	2008	103	anime	Ghibli

title	**lengthHrs**
Ponyo	**1.716**

Projection is the operation of producing an output table with tuples that have a subset of their prior attributes

Simple SQL Query: Projection

- Can even allow a constant as an expression

```
SELECT title, 'yes' AS isMovie
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

title	year	length	genre	studioName
Ponyo	2008	103	anime	Ghibli

⇩

title	**isMovie**
Ponyo	**yes**

<u>Projection</u> is the operation of producing an output table with tuples that have a subset of their prior attributes

Notation

Input schema

Movies(title, year, length, genre, studioName)

```
SELECT title, 'yes' AS isMovie
FROM Movies
WHERE studioName = 'Ghibli'
AND year = 2008;
```

Output schema

Answer(title, isMovie)

Simple SQL Query: Selection

In the WHERE clause, we may build expressions using:

- Comparison: =, <>, < , >, <=, and >=
- Arithmetic: +, -, *, /, %
- Strings: surrounded by single quotes
- Boolean operators: AND, OR, NOT

Selection is the operation of filtering a relation's tuples on some condition

```
SELECT title
FROM Movies
WHERE studioName = 'Ghibli'
AND (year > 2000 OR length <= 100);
```

Comparison of strings

- Two strings are equal if they have the same sequence of characters
  - Ignore pad characters in fixed-length CHAR(n) strings

- <, >, <=, >= comparisons are based on lexicographic order
  - ‘fodder’ < ‘foo’
  - ‘bar’ < ‘bargain’

A Few Details

- SQL **commands** are case insensitive:
  - Same: SELECT, Select, select
  - Same: Product, product
- **Values** are **not:**
  - Different: 'Seattle', 'seattle'

- Use single quotes for constants:
  - 'abc' - yes
  - "abc" - no

LIKE: Simple String Pattern Matching

```
SELECT title
FROM Movies
WHERE title LIKE 'Star ____';
```

```
SELECT title
FROM Movies
WHERE title LIKE 'Star%';
```

- s **LIKE** p: pattern matching on strings
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

DISTINCT: Eliminating Duplicates

```
SELECT DISTINCT Category
FROM   Product
```

Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

ORDER BY: Sorting the Results

```
SELECT title
FROM Movies
WHERE studioName = ‘Ghibli’
ORDER BY length, title DESC;
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

In-class Activity

- MAKE A COPY of the Notebook
- Complete the ”Set Up” section for single-table queries
- Complete Q1 and Q2

https://tinyurl.com/37x85w2j

3. Multi-table Queries

Queries involving multiple relations

- Until now, we studied queries for a single relation
- We can also combine multiple relations
  - joins, products, unions, intersections, and differences
- Why store data in multiple relations?
  - Single table
    - Data exchange is easier
    - Avoids cost of joining
  - Multiple tables
    - Data updates are easier
    - Querying a table is faster

Foreign Key constraints

Suppose we have the following schema:

Students(<u>sid</u>: string, name: string, gpa: float)

Enrolled(<u>student_id</u>: string, <u>cid</u>: string, grade: string)

And we want to impose the following constraint:

- <u>'Only bona fide students may enroll in courses'</u> i.e. a student must appear in the Students table to enroll in a class

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	564	A
123	537	A+

student_id alone is not a key - what is?

We say that student_id is a <u>foreign key</u> that refers to Students

Declaring Foreign Keys

Students(<u>sid</u>: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

```
CREATE TABLE Enrolled(
        student_id CHAR(20),
        cid              CHAR(20),
        grade   CHAR(10),
        PRIMARY KEY (student_id, cid),
        FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

Foreign Keys and update operations

Students(<u>sid</u>*: string,* name: *string*, gpa: *float*)

Enrolled(<u>student_id</u>: *string, <u>cid</u>*: *string*, grade: *string*)

What if we insert a tuple into Enrolled, but no corresponding student?

- INSERT is rejected (foreign keys are <u>constraints</u>)!

What if we delete a student?

DBA chooses (syntax in the book)

1. Disallow the delete
2. Remove all of the courses for that student
3. Set the foreign key columns to NULL (if the column is nullable)

Exercise

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: What is a foreign key vs. a key here?

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Joins

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

Joins

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

A join between tables returns all unique combinations of their tuples which meet some specified join condition

Joins

```
Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)
```

Several equivalent ways to write a basic join in SQL:

```
SELECT  PName, Price
FROM    Product, Company
WHERE   Manufacturer = CName
        AND Country='Japan'
        AND Price <= 200
```

```
SELECT  PName, Price
FROM    Product
JOIN    Company ON Manufacturer = CName
             AND Country='Japan'
WHERE   Price <= 200
```

Joins

Product

PName	Price	Category	Manuf
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
        AND Country='Japan'
       AND Price <= 200
```

PName	Price
SingleTouch	$149.99

Tuple Variable Ambiguity in Multi-Table

Person(<u>name</u>, address, worksfor)

Company(<u>name</u>, address)

```
SELECT DISTINCT name, address
FROM        Person, Company
WHERE       worksfor = name
```

Which "address" does this refer to?

Which "name"s??

Tuple Variable Ambiguity in Multi-Table

Person(name, address, worksfor)

Company(name, address)

Both equivalent ways to resolve variable ambiguity

```
SELECT DISTINCT Person.name, Person.address
FROM            Person, Company
WHERE    Person.worksfor = Company.name
```

```
SELECT DISTINCT p.name, p.address
FROM            Person p, Company c
WHERE    p.worksfor = c.name
```

Meaning (Semantics) of SQL Queries

```
SELECT x1.a1, x1.a2, …, xn.ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions(x1,…, xn)
```

$$\texttt{SELECT } x_1.a_1,\ x_1.a_2,\ \ldots,\ x_n.a_k \quad \texttt{FROM } R_1 \texttt{ AS } x_1,\ R_2 \texttt{ AS } x_2,\ \ldots,\ R_n \texttt{ AS } x_n \quad \texttt{WHERE } \text{Conditions}(x_1,\ldots,x_n)$$

Almost never the fastest way to compute it!

Answer = {}
for x_1 in R_1 do
    for x_2 in R_2 do
        …..
            for x_n in R_n do
                if Conditions(x_1,…, x_n)
                    then Answer = Answer $\cup$ {($x_1.a_1$, $x_1.a_2$, …, $x_n.a_k$)}
return Answer

Note: this is a *multiset* union

An example of SQL semantics

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

A
1
3

B	C
2	3
3	4
3	5

Cross Product

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Output

A
3
3

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

Note the *semantics* of a join

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections** / conditions:

$$Y = \{(r, s) \in X \mid r.A == r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,) \text{ for } y \in Y$$

= Returning only some attributes

Remembering this order is critical to understanding the output of certain queries

Note: we say “semantics” not “execution order”

The previous slides show *what a join means*

Not actually how the DBMS executes it under the covers

- We will discuss the execution in a later lecture

In-class Activity Continued

- Make a copy of the Collab Notebook
- Complete the Setup for multi-table queries and Q3

https://tinyurl.com/37x85w2j