Querying Databases

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain why we learn about querying a database.
- Define what a query language is, and state its important aspects.

Why Do We Build a Database?

- *So we can capture the world.*
- Then why do we capture the world?
- *So we can keep it up to date.*
- Then why do we keep it up to date?
- *So we can support our applications.*
- Then how does a database support our applications?

- It lets us **ask questions** (that our applications need)!

DBMS Is for …

- Storing information
- Organizing information
- Modifying information

- Ultimately: **Searching for** information
- Our “computing on data” is also for answering questions.
  - Relational algebra is therefore an underlying computation framework for answering questions.

How Do We Query Data?

- You can write a program to open, read, close files.
  - f = open(’datafile', ’r')
  - f.seek(120)
  - f.read(16)
  - f.close()
- That is hard to use.
- That can be quite inefficient.

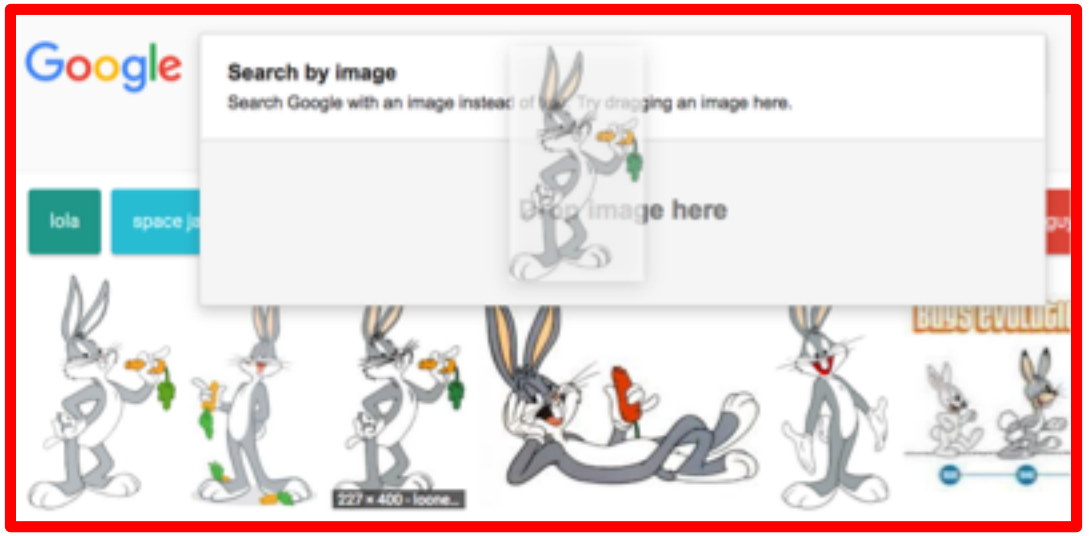
Query Languages

- Query language: A “computer language” to communicate with a database for asking questions on the information stored within.

- The design of a query language depends on
  - (Physical) Data model: How is data organized in the database?
  - Query capabilities: What types of questions can be asked?
  - Target users: Who will use the query language?
    - Programmers?
    - Non-programmers?
    - Everyday users?

Query Language: An Example

- Query language: ”Query by example”
  - Various forms of QBE were developed.
  - E.g., “Search by image” in web search.



Screenshot of “search by image” at Google image search

- Data model: Collection of images.
- Query capabilities: Finding similar images.
- Target users: Everyday users.

Food for Thought

What is the most popular query language today? Can you identify its data model, query capabilities, and target users?

Querying in the Relational Way: SQL

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

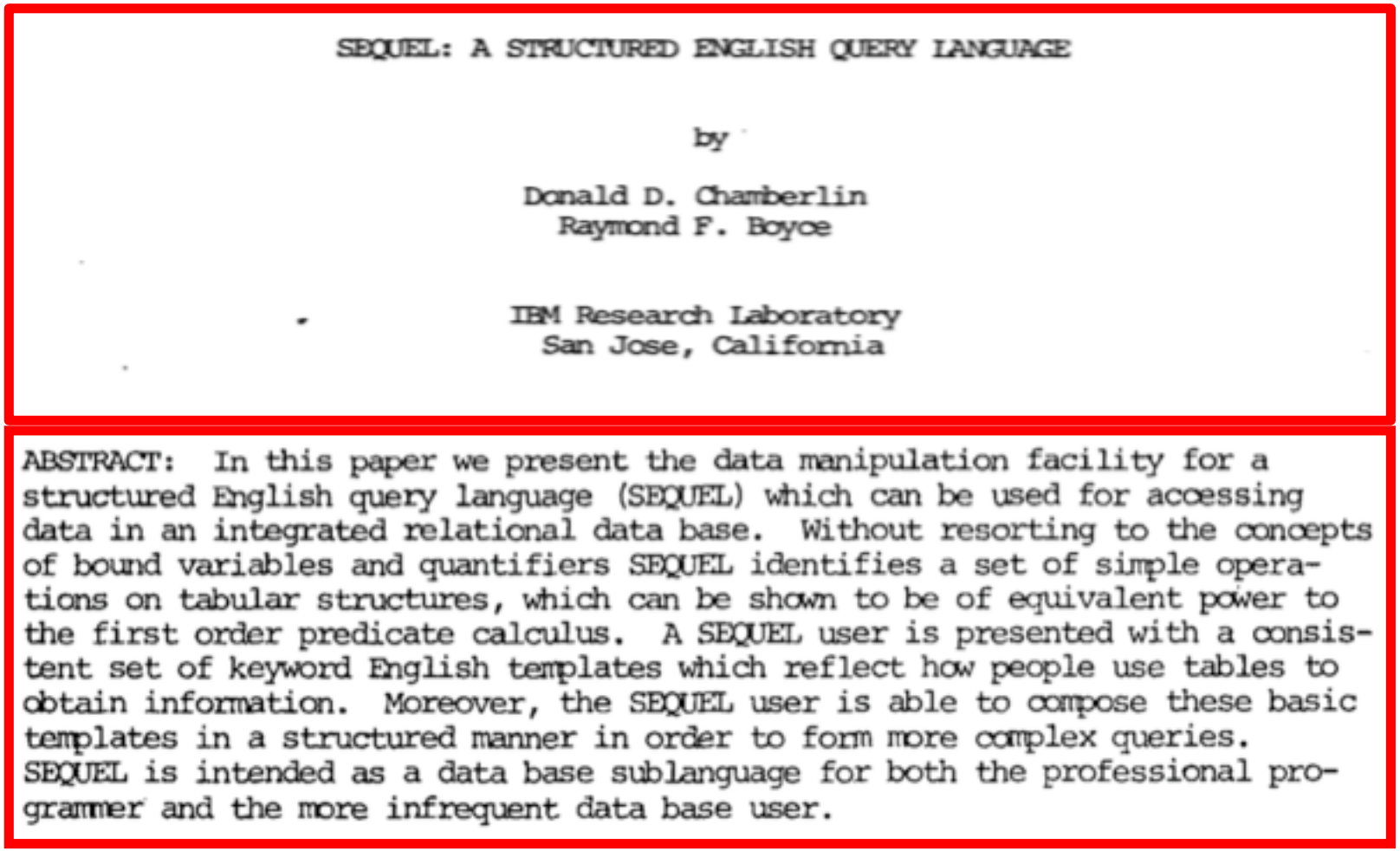
- Pronounce SQL with confidence.
- Describe the objective of SQL and its target users.
- Contrast declarative vs. procedural languages.
- Explain why SQL can be both easy to use and efficient to execute.

SQL Introduction

- **S**tructured **Q**uery **L**anguage.
- Pronounced “S-Q-L” or “sequel”.
- Standard language for querying and manipulating data.

- Created at IBM, since 1970’s.
- Standardized in 1986. Many revisions were made since then:
  - SQL-86, SQL-89, SQL-92, SQL-1999, SQL:2003, …, SQL:2016.
- Vendors support various subsets of these.
- We will introduce the essential that most vendors support.

From the Inventors: SQL - What, Why, for Whom?



SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE

by

Donald D. Chamberlin
Raymond F. Boyce

IBM Research Laboratory
San Jose, California

ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.

- *Chamberlin, D. D. and Boyce, R. F. 1974. SEQUEL: A structured English query language. In Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (Ann Arbor, Michigan, May 01 - 03, 1974). FIDET '74. ACM, New York, NY, 249-264.*

SQL Is Declarative: What, Not How!

- SQL is a very-high-level language: It is **declarative**.
  - Using a “consistent set of keyword English templates
- You only say **what** you want, and not **how** to get it.
  - Unlike in “high-level” programming languages like C++ or Python.

Declarative:

Can It Be Both Easy and Efficient?

- Ease of Use: SQL targets at both the professional programmer and the more infrequent data base users.

> A brief discussion of this new class of users is in order here. There are some users whose interaction with a computer is so infrequent or unstructured that the user is unwilling to learn a query language. For these users, natural language or menu selection (3,4) seem to be the most viable alternatives. However, there is also a large class of users who, while they are not computer specialists, would be willing to learn to interact with a computer in a reasonably high-level, non-procedural query language. Examples of such users are accountants, engineers, architects, and urban planners. It is for this class of users that SEQUEL is intended. For this reason, SEQUEL emphasizes simple data structures and operations.

Excerpt from the SEQUEL paper

- Efficiency:
  - Even though programmers are smart (we know you are), database is dynamic-- data can change!
  - Declarative queries can be “optimized” by system “just in time”.

The First Ever SQL Queries?

- First SQL query:

As in SQUARE, the simplest SEQUEL expression is a mapping which specifies a table, a domain, a range, and an argument, as illustrated by Q1.

Q1. Find the names of employees in the toy department.

```
SELECT    NAME
FROM      EMP
WHERE     DEPT = 'TOY'
```

The mapping returns the entire set of names which qualify according to the test DEPT = 'TOY'.

- (Bonus) First SQL join query:

name. This is illustrated by Q10, which implements what Codd (7) would call a "join" between the SALES and SUPPLY tables on their ITEM columns:

Q10. List rows of SALES and SUPPLY concatenated together whenever their ITEM values match.

```
SALES, SUPPLY
WHERE SALES . ITEM = SUPPLY . ITEM
```

Food for Thought

What "example scenarios" did SQL inventors assume?
What were their target application settings?

SQL: Basic Form

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Specify the basic form of an SQL query.
- Describe the SELECT-FROM-WHERE construct and what each clause means.
- Use SELECT-FROM-WHERE to write queries.

Scenario: Academic World

CREATE TABLE **Students** (
name text,
major text,
year text,
advisor text)

CREATE TABLE **Professors** (
name text,
dept text,
course text)

CREATE TABLE **Courses** (
number text,
title text,
credit integer)

CREATE TABLE **Enrolls** (
name text,
number text,
grade integer)

Example Application 1: *Academic World*

Academic World

- What you do in an academic world.
- Who they are: students, courses, and professors.
- What they do: students taking courses, professors teaching courses and advising students.

Academic world (Munroe, n.d.)

Scenario: Friday Night

CREATE TABLE **Drinkers** (
name text,
addr text,
hobby text,
frequent text)

CREATE TABLE **Bars** (
name text,
addr text,
owner text)

CREATE TABLE **Beers** (
name text,
brewer text,
alcohol real)

CREATE TABLE **Sells** (
bar text,
beer text,
price real,
discount real)

CREATE TABLE **Drinks** (
drinker text,
beer text,
rating integer)

CREATE TABLE **Favorites** (
drinker text,
bar text,
beer text,
season text)

Example Application 2: *Friday Night*

Friday Night

- What *some* probably do on a Friday night.
- Who they are: drinkers, beers, and bars.
- What they do: drinkers frequenting bars and drinking favorite beers, bars selling beers.

Beer mug

Scenario: Friday Night

Basic Form: SELECT-FROM-WHERE

- The SELECT-FROM-WHERE from:

  SELECT $A_1, \ldots, A_n$ ← Attributes to return.

  FROM $R_1, \ldots, R_m$ ← Relations to query from.

  WHERE Condition ← Condition that results must satisfy.

Basic-Form Query Examples

- *Q1: Find students' name and year for those who major in Econ.*
- *Q2: Find the courses that are taught by CS professors.*
- *Q3: Find the prices and bars to get the Sam Adams beer.*

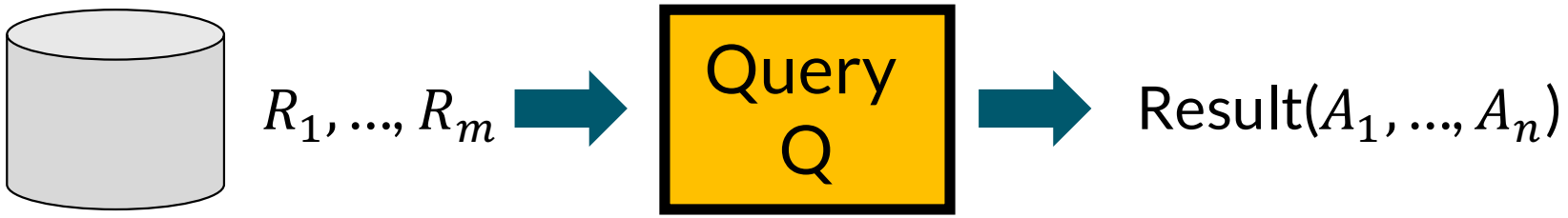
Q1
Select name, year from students where major = “Econ”

Q2
Select students.name, enrolls.grade from students.enroll where students.name = enrolls.name and enrolls.number = “CS411”

Q3
Select price, bar from sells where beer = “Sam Adams”

SQL Query: Relations In, Relation Out

- Input: Relations R_1, …, R_m (a subset of relations in DB)
- Output: Relation with attributes A_1, …, A_n
  - No name-- you can call it the "Result" relation.



From database to query result

- Closure property:
  - SQL, like relational algebra is closed with respect to the relational model.
  - Each operation takes one or more relations and returns a relation.
- We can thus use a **query as a relation.**
  - Anywhere when a relation is expected.
  - E.g., the concept of "view".

What does it mean?

- Combine (Cartesian product) the relations in the FROM clause.
- Filter it by the condition in the WHERE clause.
- Output the tuples with the attributes in the SELECT clause.

- In terms of relational algebra:

$$\pi_{A_1,\ldots,A_n}(\sigma_C\ (R_1 \times \cdots \times R_m))$$

FROM Clause

- Specify one or more “source” relations in DB: **FROM** R_1, …, R_m
- Tuple Variable: You can use an “alias” to name a relation.
  - For referencing in WHERE and SELECT.
  - Think of it as representing a tuple in the relation.
- Useful for multiple reasons
  - To use as a short name:
    - FROM Favorites F, …
  - To distinguish multiple instances of the same relation.
    - FROM Drinkers D1, Drinkers D2

FROM-Clause Query Examples

- *Q1: Find the names, majors, and grades for those students who took CS411.*

- *Q2: Find those drinkers who share the same favorite beers.*

Q1
Select s.name, s.major, e.grade from student s, enrolls e where s.name = e.name and e.number = “CS411”

Q2
Select f1.drinker, f2.drinker, f1.beer from favorites f1, favorites f2 where f1.beer = f2.beer and f1.drinker < f2.drinker

Referring to an Attribute

- If no ambiguity, simply use its name.

- If ambiguity (or simply for clarify/readability)
  - <relation>.<attribute>, e.g., Drinkers.name, Bars.name.
  - <tuple variable>.<attribute>, e.g., F.beer, D1.name, D2.name.

SELECT Clause

- Specify each attribute: **SELECT** A_1, …, A_n

- Get all attributes: SELECT *
- Rename attributes: SELECT …, attr AS new_name, …
- Transform attributes by expression: SELECT …, expression AS name, …
  - An **expression** is a combination of one or more values, operators, and SQL functions that evaluates to a value.

SELECT-Clause Query Examples

- *Q1: Find the beers and their actual prices sold at Green Bar.*

Q1
Select beer, price * (1 - discount) AS actual_price from sells where bar = “green bar”

WHERE Clause

- WHERE C
- C can be
  - simple condition, or
  - complex condition consisting of multiple simple conditions connected with Boolean operators AND, OR, NOT and parentheses.

Conditions in WHERE-Clause

- Attributes: from the relations in FROM.
  - Prefixed by relation name or tuple variable to uniquely reference.
- Arithmetic operations.
  - price * discount < 1.0.
- String operations
  - UPPER(bar) = ‘GREEN BAR’
- Comparison operators: =, <>, <, >, <=, >=.
- Pattern matching: LIKE operator: bar LIKE ’%Green%’
  - % = “any string”; _ = “any character.”

WHERE-Clause Query Examples

- *Q1: Find the beers whose actual prices are less than $3.5.*

- *Q2: Find the courses whose numbers start with “CS”.*

Q1
Select beer, price from sells where price * (1 - discount) < 3.5

Q2
Select number, title from course where number like “CS%”

Food for Thought

SEQUEL was not the only one-- there was QUEL from the INGRES project in the dawn of the relational age. ***See similar ideas?***

The Design and Implementation of INGRES

MICHAEL STONEBRAKER, EUGENE WONG, AND PETER KREPS
University of California, Berkeley
and
GERALD HELD
Tandem Computers, Inc.

The currently operational (March 1976) version of the INGRES database management system is described. This multiuser system gives a relational view of data, supports two high level nonprocedural data sublanguages, and runs as a collection of user processes on top of the UNIX operating system for Digital Equipment Corporation PDP 11/40, 11/45, and 11/70 computers. Emphasis is on the design decisions and tradeoffs related to (1) structuring the system into processes, (2) embedding one command language in a general purpose programming language, (3) the algorithms implemented to process interactions, (4) the access methods implemented, (5) the concurrency and recovery control currently provided, and (6) the data structures used for system catalogs and the role of the database administrator.

Also discussed are (1) support for integrity constraints (which is only partly operational), (2) the not yet supported features concerning views and protection, and (3) future plans concerning the system.

The following suggest valid QUEL interactions. A complete description of the language is presented in [15].

Example 1.1. Compute salary divided by age-18 for employee Jones.

```
RANGE OF E IS EMPLOYEE
RETRIEVE INTO W
(COMP = E.SALARY/(E.AGE-18))
WHERE E.NAME = "Jones"
```

Example 1.3. Fire everybody on the first floor.

```
RANGE OF E IS EMPLOYEE
RANGE OF D IS DEPT
DELETE E WHERE  E.DEPT = D.DEPT
          AND  D.FLOOR# = 1
```

Michael Stonebraker, Eugene Wong, Peter Kreps, Gerald Held. The Design and Implementation of INGRES. ACM Trans. Database Syst. 1(3): 189-222 (1976)

SQL: Null Values

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe what NULL values are, and why they appear in databases.
- Identify the truth value of conditions that involve NULL values.

What Should This Query Return?

```
SELECT bar, beer
FROM Sells
WHERE price < 5.00  OR  price >= 5.00;
```

vs.

```
SELECT bar, beer FROM Sells;
```

Null Values

- Tuples in SQL relations can have NULL as a value for one or more components.

- Meaning depends on context.
- Two common cases:
  - Missing value :
    - E.g., we don’t know the exact price of a beer.
  - Inapplicable :
    - E.g., the grade of a no-grading class.

Comparing NULL's to Values

- When NULL is compared with any value, the truth value is UNKNOWN.
- Thus, SQL needs to employ 3-valued logic: TRUE, FALSE, UNKNOWN.
- But a query only produces a tuple in the answer if its truth value for the WHERE clause is TRUE (not FALSE or UNKNOWN).

3-Valued Logic Involving Unknown

- AND: TRUE if both are TRUE.
  - UNKNOWN AND TRUE = UNKNOWN
  - UNKNOWN AND FALSE = FALSE
  - UNKNOWN AND UNKNOWN = UNKNOWN
- OR: TRUE if either is TRUE.
  - UNKNOWN OR TRUE = TRUE
  - UNKNOWN OR FALSE = UNKNOWN
  - UNKNOWN OR UNKNOWN = UNKNOWN
- NOT: Just flip it.
  - NOT UNKNOWN = UNKNOWN

The Surprise Explained

```
SELECT bar, beer
FROM Sells
WHERE price < 5.00  OR  price >= 5.00;
```

- For a tuple with price = NULL:
  - price < 5.0 = UNKNOWN.
  - price >= 5.0 = UNKNOWN.
  - UNKNOWN OR UNKNOWN = UNKNOWN.
  - Thus this tuple is not in the result.

Testing for NULL

- To test and include/exclude NULL:
  - x IS NULL.
  - x IS NOT NULL.

```
SELECT bar, beer
FROM Sells
WHERE price < 5.00  OR  price >= 5.00 OR price IS NULL;
```

Null-Value Query Examples

- *Q1: Find all beers regardless of their price comparison to $5.*

- *Q2: Find the beers whose prices are unknown.*

Q1
select beer, price from sells where price < 5 or price >= 5 or price is null

Q2
select beer, price from sells where price is null

SQL: Subqueries

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Define what subqueries are, and where they can appear in a query.
- Identify different types of subqueries that return a value or a relation, and explain their differences.
- Use operators that compare relations (returned by subqueries).
- Write a query with subqueries.

Subquery

- A parenthesized SELECT-FROM-WHERE contained in a query.
- Can contain subqueries.
- Can return a relation or a single constant value.
- Can be used in any place when relation or constant values are expected- FROM, WHERE, and even SELECT.

Subqueries that Return Single Constants

- If a subquery is guaranteed to produce **one tuple with one attribute**, then it can be used as a scalar value.
  - Note it is still a relation, just a special one.
- Can be used anywhere a scalar value is expected.
  - WHERE -- Our focus.
  - SELECT too!
- ”Single constant” can be guaranteed by:
  - Key constraints
    - E.g., Enrolls(name, number, grade): One grade for a given name and number.
  - Aggregate functions
    - E.g., Avg(price) of all Sells tuples.

Scalar-Subquary Examples

- *Q1: Find those students who has a better grade than Bugs Bunny in CS411.*

- *Q2: Find the bars that serve Sam Adams cheaper than the average market price.*

Q1
select title from courses where (number) in
(
select number from enrolls where name = “bugs bunny”
)

Q2
select bar, beer from sells s1 where not exists
(
select bar, beer from sells s2 where s2.beer = “Bud” and s2.bar <> “sober bar”
)

Subqueries that Return Relations

- In general, a subquery (like any query) would return a relation.

- Can be used anywhere a relation is expected.
  - WHERE -- Our focus.
  - FROM too!

Using Relation-Subquaries in WHERE Conditions

- Relation-subquery returns a relation-- i.e., a set of tuples.
- How to use a relation in a condition?

- We need operators that deal with sets.

Operators That Deal with a Relation

- IN (or NOT IN)
  - **<tuple> IN <relation>:** if the tuple is a member of the relation.
- EXISTS (or NOT EXISTS)
  - **EXISTS <relation>:** if the <relation> is not empty.
- ANY
  - ***X* <OP> ANY <relation>:** *X* OP (=, <=, etc.) at least one tuple in the relation.
  - <relation> has only one attribute (so each tuple is a single value).
- ALL
  - ***X* <OP> ALL <relation>:** *X* OP (=, <=, etc.) all tuples in the relation.

Relation-Subquery Examples

- *Q1: Find the title of courses that Bugs Bunny take. Use IN.*
- *Q2: Find the bars that sell unique beers (that no other bars have). Use EXISTS.*
- *Q3: Find the students who take a class taught by their advisers. Use ANY. (Note: ANY is not supported in RASQL, which uses SQLite.)*
- *Q4: Find the most expensive beer on the market. Use ALL. (Note: ALL is not supported in RASQL, which uses SQLite.)*

Food for Thought

Can you think of a query that you can ask with IN or EXISTS or ANY or ALL-- in four different forms?

SQL: Set Operations

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Identify set operators used in SQL.
- Write queries with set operations.

Union, Intersection, and Difference

- Union
  - (subquery) UNION (subquery)
- Intersection
  - (subquery) INTERSECT (subquery)
- Difference
  - (subquery) EXCEPT (subquery)

Set Operation Examples

- *Q1: Find the ”really-happy drinkers” who have bars on the same streets they live and the bars sell beers that they drink.*

- *Q2: Find the ”probably-happy drinkers” who have bars on the same streets they live or bars that sell beers that they drink.*

Q1
select d.name, b.name from drinkers d, bars b where d.addr = b.addr

Q2
select d.name, b.name from drinkers d, bars b where d.addr = b.addr
intersect
select d.drinker, s.bar from drinks d, sells s where d.beer = s.beer

SQL: Dealing with Duplicates

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain why relations may contain duplicates.
- Define set vs. bag semantics for SQL queries.
- Identify the default semantics in SQL queries.
- Use operators to override default and control duplicates.

Duplicates: Set or Bag for Your Data?

- Set:
  - A **set** is a collection of distinct elements.
  - No duplicates. Multiplicity does not count.
  - $\{a, a, b\} = \{a, b\}$
- Bag (or multiset):
  - A **bag** is a collection of **multiple instances of** distinct elements.
  - Duplicates allowed. Multiplicity counts.
  - $\{a, a, b\} \neq \{a, b\}$

- For a relation stored in your DB: Perhaps no duplicates.
  - Since real-world entities are supposed to be unique.
  - You can use “key” attributes to enforce uniqueness (e.g., “id” must be different).

Would You Want Duplicates? Sometimes!

- (Intermediate) results may not be distinct entities.
- Their multiplicity may carry information you want!

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

- *Q1: What majors are there?*
  - {CS, Bio, Econ}
- *Q2: What major is "most popular"?*
  - {CS x 2, Bio x 1, Econ x 1} → CS most popular

SQL: Default about Duplicates

- SELECT-FROM-WHERE: Default is BAG semantics.
  - Why? A query generate results (and not stored relations), the multiplicity probably carries information we want.
- UNION/INTERSECT/EXCEPT: Default is SET semantics.
  - Why? Because these are set operations!
- In either case, you can override defaults.

Overriding Default Semantics

- SELECT-FROM-WHERE: Bag semantics
- Override with “DISTINCT”
  - SELECT major → {CS, Bio, Econ, CS}
  - SELECT DISTINCT major → {CS, Bio, Econ}

- Set Operations: Set semantics
- Override with “ALL”
  - (subquery) UNION (subquery)
  - (subquery) UNION ALL (subquery)

Handling-Duplicates Query Examples

- *Q1: Find those beers that are sold on the market.*

- *Q2: Find all the grades of CS411 and CS423 (in order to take average).*

Q1
select distinct beer from sells

Q2
select grade from enrolls where number = “CS411”
union all
select grade from enrolls where number = “CS423”

SQL: Aggregates

Querying Databases: The Relational Way

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain what an aggregate query is and why we need it.
- Identify the key ingredients of an SQL aggregate query.
- Distinguish attributes for tuples vs. groups and when to use which attributes in an aggregate query.
- Write aggregate queries using GROUP-BY, HAVING, and aggregate functions.

Aggregate Query Examples

- *Q1: Find the average grade of CS411. Compare with CS423.*

- *Q2: : Find the average price of each beer brewed by "AB InBev" if it is sold at multiple bars.*

Q1
select avg(grade) from enrolls where number = “CS411”
select avg(grade) from enrolls where number = “CS423”

Q2
select beer, bar, avg(price), count(bar) from beers, sells
where beers.name = sells.beer and brewer = “AB InBev”
group by beer

What is Aggregation? Why Need It?

- Aggregate function: E.g., SUM, AVG, MIN, COUNT
  - Input: a collection of values
  - Output: a value

- Why? We want to see the "big picture".
  - Summary of data.
  - Every element contributes!
    - Even MIN and MAX are the results of everyone.

```
SELECT AVG(grade)
FROM Enrolls
WHERE number = "CS411"
```

Example aggregate query

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

SQL Aggregation Framework: Key Ingredients

- **1) FROM-WHERE** to generate tuples
- **2) GROUP-BY** to organize groups
- **3) Functions** to aggregate each group
- **4) HAVING** to filter groups
- **5) SELECT** attributes of groups to output

SELECT $A_1, \ldots, A_k, F(A_i)$
FROM $R_1, \ldots, R_m$
WHERE C
GROUP BY $A_1, \ldots, A_k$
HAVING H

Standard form of aggregate queries

Q: Find the average prices of beers that are sold at multiple bars.

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

1) FROM-WHERE to Generate Tuples

- An aggregate needs to apply upon a collection of tuples.
- Where do these tuples come from?
- We specify these tuples by FROM R_1, …, R_m WHERE …

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Bud	AB InBev	5
Bud Lite	AB InBev	4.2
Coors	Coors	5

bar	beer	price
Sober Bar	Bud	5
Sober Bar	Bud Lite	3
Sober Bar	Sam Adams	6
Green Bar	Sam Adams	4.5
Green Bar	Bud	2
Green Bar	Coors	
Purple Bar	Sam Adams	5

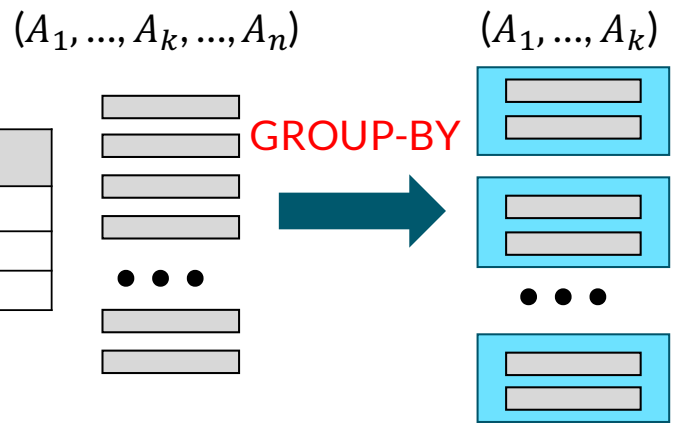
name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	Bud	5
Bud	AB InBev	5	Green Bar	Bud	2
Bud Lite	AB InBev	4.2	Sober Bar	Bud Lite	3

Aggregate query framework

2) GROUP-BY to Organize Groups

- Aggregates are applied to a collection of tuples.
- What would be “a collection” to apply to?
- We can specify attributes of tuples to organize them together.
- **GROUP-BY A_1, …, A_k** (subset of attributes from $R_1 \times \cdots \times R_m$)
  - Tuples sharing the same A_1, …, A_k values will be in the same group.
  - Thus, each group is represent by attributes A_1, …, A_k.

name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	Bud	5
Bud	AB InBev	5	Green Bar	Bud	2
Bud Lite	AB InBev	4.2	Sober Bar	Bud Lite	3



name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	**Bud**	5
Bud	AB InBev	5	Green Bar		2
Bud Lite	AB InBev	4.2	Sober Bar	**Bud Lite**	3

Aggregate query framework

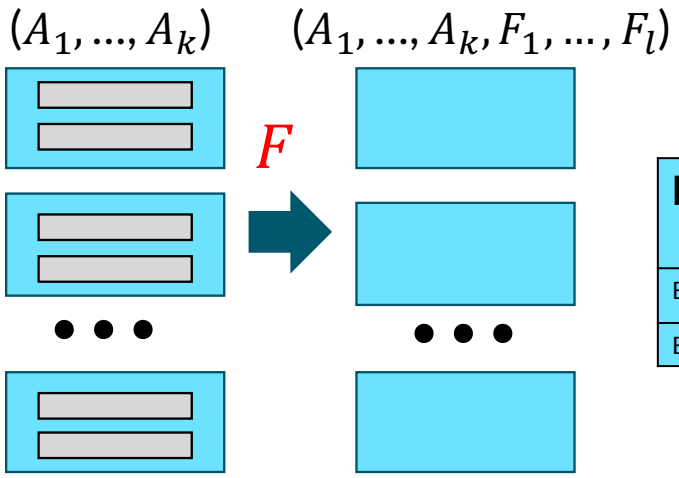
3) Functions to Aggregate Each Group

- Applying to Attributes
  - SUM, AVG, COUNT, MIN, and MAX.
- Applying to Tuples
  - COUNT(*) counts the number of tuples.

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	**Bud**	5
Bud	AB InBev	5	Green Bar		2
Bud Lite	AB InBev	4.2	Sober Bar	**Bud Lite**	3



Aggregate query framework

beer	AVG (price)	COUNT (bar)
Bud	3.5	2
Bud Lite	3	1

Aggregates: What Members Count?

- Null values
  - Null does not contribute to aggregation.
  - If there are no non-Null values in a column, then the result of the aggregation is Null(for average, sum) or 0 (for count).

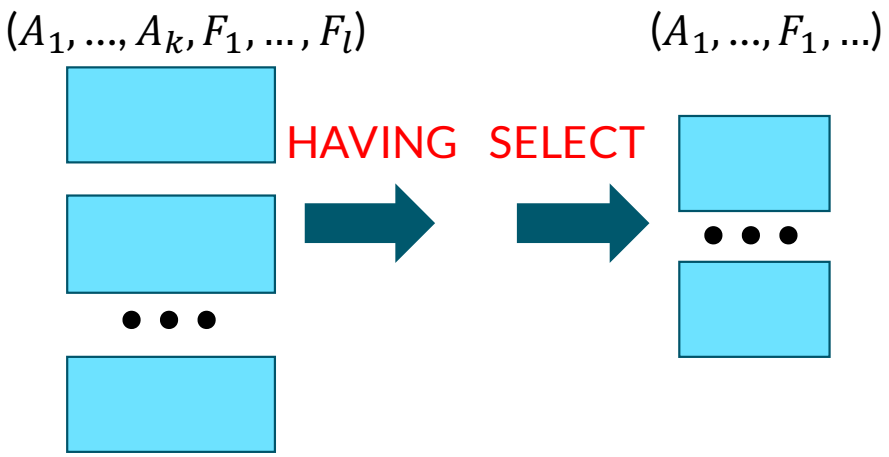
- Duplicates
  - DISTINCT inside an aggregation causes duplicates to be eliminated before the aggregation.
  - E.g., SELECT COUNT(DISTINCT price)

4) HAVING to Filter Groups

- HAVING <condition *H*>
- The conditions refers to “group attributes”.
  - GROUP-BY attributes, or
  - Aggregate functions.
- *H* applies to each group, and groups not satisfying the condition are eliminated.

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = “AB InBev”
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query



Aggregate query framework

beer	AVG (price)	COUNT (bar)
Bud	3.5	2
Bud Lite	3	1

beer	AVG (price)
Bud	3.5

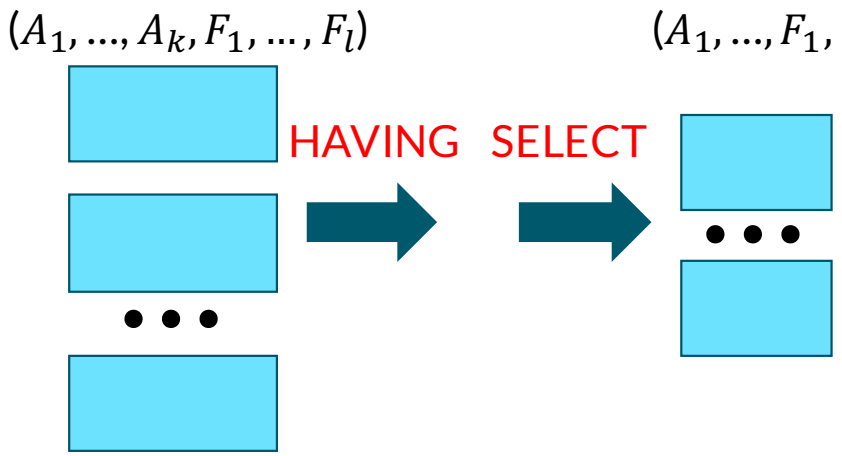
5) SELECT Attributes of Groups to Output

- SELECT $A_1, \ldots, A_k, F(A_i)$
  - Using only attributes or aggregates of each group.

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

beer	AVG (price)	COUNT (bar)
Bud	3.5	2
Bud Lite	3	1



Aggregate query framework

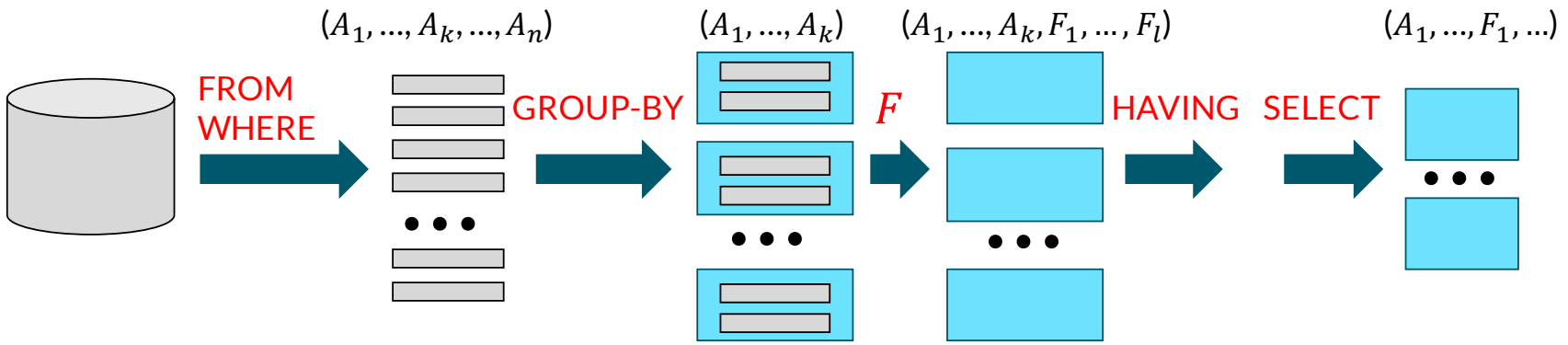
beer	AVG (price)
Bud	3.5

SQL Aggregate: The Overall Framework

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Bud	AB InBev	5
Bud Lite	AB InBev	4.2
Coors	Coors	5

bar	beer	price
Sober Bar	Bud	5
Sober Bar	Bud Lite	3
Sober Bar	Sam Adams	6
Green Bar	Sam Adams	4.5
Green Bar	Bud	2
Green Bar	Coors	
Purple Bar	Sam Adams	5

name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	**Bud**	5
Bud	AB InBev	5	Green Bar		2
Bud Lite	AB InBev	4.2	Sober Bar	**Bud Lite**	3



beer	AVG (price)
Bud	3.5

name	brewer	alcohol	bar	beer	price
Bud	AB InBev	5	Sober Bar	Bud	5
Bud	AB InBev	5	Green Bar	Bud	2
Bud Lite	AB InBev	4.2	Sober Bar	Bud Lite	3

beer	AVG (price)	COUNT (bar)
Bud	3.5	2
Bud Lite	3	1

Aggregate query framework

Attributes of Tuples vs. Groups

- In an aggregate query, there are tuples and groups (of tuples).
- They have different sets of attributes.

- **FROM-WHERE-GRPUP-BY: Using attributes of tuples.**
- **HAVING and SELECT: Using attributes of groups**

```
SELECT beer, AVG(price)
FROM Beers, Sells
WHERE Beers.name = Sells.beer
      and brewer = "AB InBev"
GROUP BY beer
HAVING COUNT(bar) >= 2
```

Example aggregate query

Group-By Is Optional: Aggregates without Explicit GROUP-BY

- One Group containing all tuples generated from FROM-WHERE

```
SELECT AVG(grade) FROM Enrolls WHERE number = “CS411”
```

- No GROUP-BY
  - The group has NO common “group-by” attributes.
  - Thus, SELECT can only use aggregate functions.

So, These Queries Are Illegal!

- Why? They use “non-group” attributes after grouping.

- With explicit GROUP-BY:
  - SELECT beer, MIN(price) *// Or, AVG(price), MAX(price) -- any aggregates.*
    FROM Beers, Sells
    GROUP BY bar

- Without explicit GROUP-BY:

  - SELECT beer, AVG(price) *// Or, MIN(price), COUNT(price) -- any aggregates.*
    FROM Sells

Food for Thought

This query is illegal in SQL. Do you agree?
What do you think it intends to ask?
How do you ask the question "correctly"?

```
SELECT name, MAX(grade)
FROM Enrolls
WHERE number = "CS411"
```

Food for Thought

This aggregate query was written by the inventors of SQL in their original paper. Did you see anything wrong?

If mathematical functions appear in the expression, their argument is taken from the set of rows of the table which qualify by the WHERE clause. For example:

Q4.1. List each employee in the shoe department and his deviation from the average salary of the department.

```
SELECT    NAME, SAL - AVG (SAL)
FROM      EMP
WHERE     DEPT = 'SHOE'
```

Example query from the original SEQUEL paper

Chamberlin, D. D. and Boyce, R. F. 1974. SEQUEL: A structured English query language. In Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control (Ann Arbor, Michigan, May 01 - 03, 1974). FIDET '74. ACM, New York, NY, 249-264.