

Lectures 2&3: Introduction to SQL

Lecture 2: SQL Part I

Today's Lecture

1. SQL introduction & schema definitions
 - ACTIVITY: Table creation
2. Basic single-table queries
 - ACTIVITY: Single-table queries!
3. Multi-table queries
 - ACTIVITY: Multi-table queries!

1. SQL Introduction & Definitions

Basic SQL

SQL Introduction

- 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

NB: Probably the world's most successful **parallel** programming language (multicore?)

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

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)
```

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

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

2. Single-table queries

SQL Query

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

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

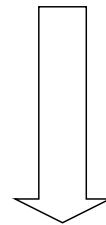
Call this a SFW query.

Simple SQL Query: Selection

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

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

```
SELECT *
FROM Product
WHERE Category = 'Gadgets'
```



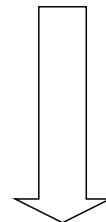
PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks

Simple SQL Query: Projection

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

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

```
SELECT Pname, Price, Manufacturer  
FROM Product  
WHERE Category = 'Gadgets'
```



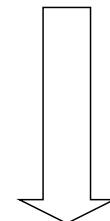
PName	Price	Manufacturer
Gizmo	\$19.99	GizmoWorks
Powergizmo	\$29.99	GizmoWorks

Notation

Input schema

Product(PName, Price, Category, Manfacturer)

```
SELECT Pname, Price, Manufacturer  
FROM Product  
WHERE Category = 'Gadgets'
```



Output schema

Answer(PName, Price, Manufacturer)

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

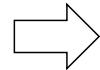
LIKE: Simple String Pattern Matching

```
SELECT *  
FROM Products  
WHERE PName LIKE '%gizmo%'
```

- $s \text{ } \text{LIKE} \text{ } p$: pattern matching on strings
- p may contain two special symbols:
 - $\%$ = any sequence of characters
 - $_$ = any single character

DISTINCT: Eliminating Duplicates

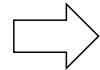
```
SELECT DISTINCT Category  
FROM Product
```



Category
Gadgets
Photography
Household

Versus

```
SELECT Category  
FROM Product
```



Category
Gadgets
Gadgets
Photography
Household

ORDER BY: Sorting the Results

```
SELECT      PName, Price, Manufacturer  
FROM        Product  
WHERE       Category='gizmo' AND Price > 50  
ORDER BY    Price, PName
```

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

Ordering is ascending, unless you specify the DESC keyword.

3. Multi-table queries

Foreign Key constraints

- Suppose we have the following schema:

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

`Enrolled(student_id: string, cid: string, grade: string)`

- And we want to impose the following constraint:

- ‘Only bona fide students may enroll in courses’ 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 foreign key that refers to Students

Declaring Foreign Keys

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

`Enrolled(student_id: string, cid: string, grade: string)`

- What if we insert a tuple into Enrolled, but no corresponding student?
 - INSERT is rejected (foreign keys are constraints)!
- What if we delete a student?
 1. Disallow the delete
 2. Remove all of the courses for that student
 3. SQL allows a third via *NULL* (*not yet covered*)

DBA chooses (syntax in the book)

Keys and Foreign Keys

Company

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

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.

*Note: we will often omit
attribute types in schema
definitions for brevity, but
assume attributes are
always atomic types*

```
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
```

A few more later on...

Joins

Product

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

Company

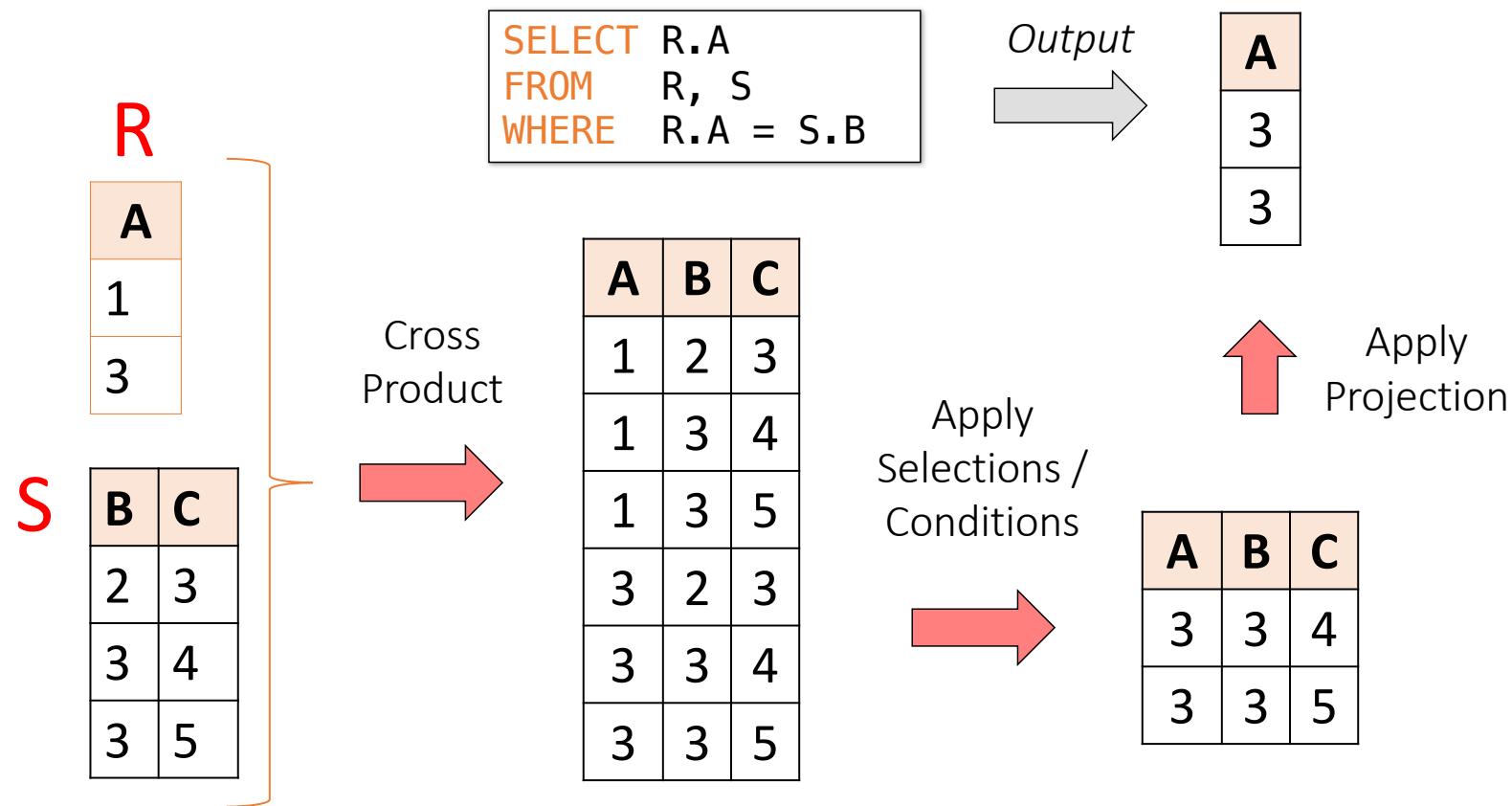
Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan



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

PName	Price
SingleTouch	\$149.99

An example of SQL semantics



Note the *semantics* of a join

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

1. Take **cross product**:

$$X = R \times S$$

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

Ex: $\{a,b,c\} \times \{1,2\}$
 $= \{(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)\}$

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 (see later on...)

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

- The preceding slides show *what a join means*
- Not actually how the DBMS executes it under the covers

A Subtlety about Joins

```
Product(PName, Price, Category, Manufacturer)  
Company(CName, StockPrice, Country)
```

Find all countries that manufacture some product
in the ‘Gadgets’ category.

```
SELECT Country  
FROM Product, Company  
WHERE Manufacturer=CName AND Category='Gadgets'
```

A subtlety about 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 Country  
FROM Product, Company  
WHERE Manufacturer=Cname  
AND Category='Gadgets'
```

Country

?

?

What is the problem ?

What's the solution ?

Lecture 3: SQL Part II

2. Aggregation & GROUP BY

Aggregation

```
SELECT AVG(price)  
FROM Product  
WHERE maker = "Toyota"
```

```
SELECT COUNT(*)  
FROM Product  
WHERE year > 1995
```

- SQL supports several **aggregation** operations:
 - SUM, COUNT, MIN, MAX, AVG

Except COUNT, all aggregations apply to a single attribute

Aggregation: COUNT

- COUNT applies to duplicates, unless otherwise stated

```
SELECT COUNT(category)
FROM Product
WHERE year > 1995
```

Note: Same as COUNT().
Why?*

We probably want:

```
SELECT COUNT(DISTINCT category)
FROM Product
WHERE year > 1995
```

More Examples

```
Purchase(product, date, price, quantity)
```

```
SELECT SUM(price * quantity)
FROM Purchase
```

What do these mean?

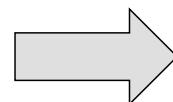
```
SELECT SUM(price * quantity)
FROM Purchase
WHERE product = 'bagel'
```

Simple Aggregations

Purchase

Product	Date	Price	Quantity
bagel	10/21	1	20
banana	10/3	0.5	10
banana	10/10	1	10
bagel	10/25	1.50	20

```
SELECT SUM(price * quantity)
FROM Purchase
WHERE product = 'bagel'
```



50 (= 1*20 + 1.50*20)

Grouping and Aggregation

Purchase(product, date, price, quantity)

```
SELECT      product,  
            SUM(price * quantity) AS TotalSales  
FROM        Purchase  
WHERE       date > '10/1/2005'  
GROUP BY    product
```

Find total sales
after 10/1/2005
per product.

Let's see what this means...

3. Advanced SQL-izing

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store  
FROM Product  
INNER JOIN Purchase  
ON Product.name = Purchase.prodName
```

Note: another equivalent way to write an
INNER JOIN!



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store  
FROM Product  
LEFT OUTER JOIN Purchase  
ON Product.name = Purchase.prodName
```



name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Other Outer Joins

- Left outer join:
 - Include the left tuple even if there's no match
- Right outer join:
 - Include the right tuple even if there's no match
- Full outer join:
 - Include the both left and right tuples even if there's no match

Summary

SQL is a rich programming language
that handles the way data is processed
declaratively