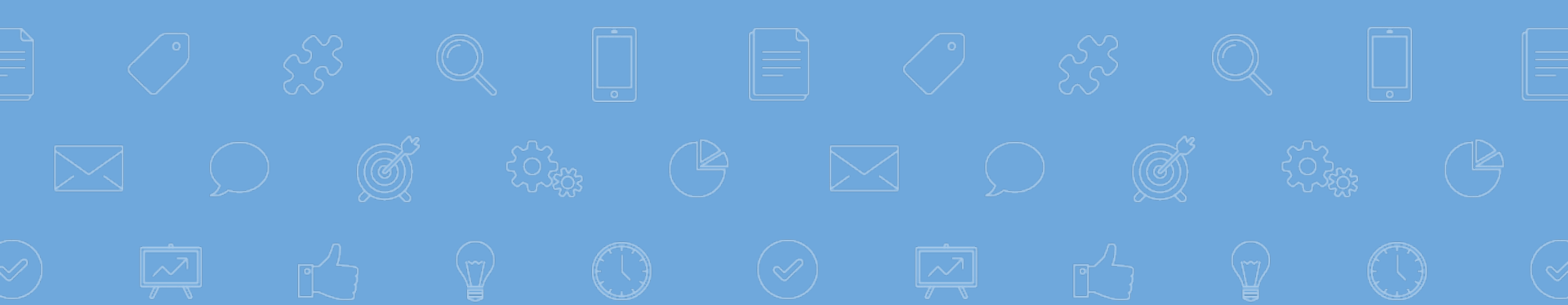


Introduction to SQL

Part I

Goals for

Today & Next week

(SQL, SQL, SQL)

Phase I: Intuition for SQL (1st half of today)

Basic Relational model (aka tables)

Example SQL (exploring real datasets)

Phase II: Formal description

SQL concepts we'll study (similar to Python map-reduce)

Schemas, Query structure of SELECT-FROM-WHERE, JOINs, etc

Data independence

Goal

Data independence

1. Can we add a new column or attribute without rewriting the application?

Logical Data Independence
Protection from changes in the Logical Structure of the data

2. Do you need to care which disks/machines are the data stored on?

Physical Data Independence
Protection from Physical Layout Changes

Key concept

Data model

Relational model (aka tables)

- Simple and most popular
- Elegant algebra (E.F. Codd et al)

Data model:

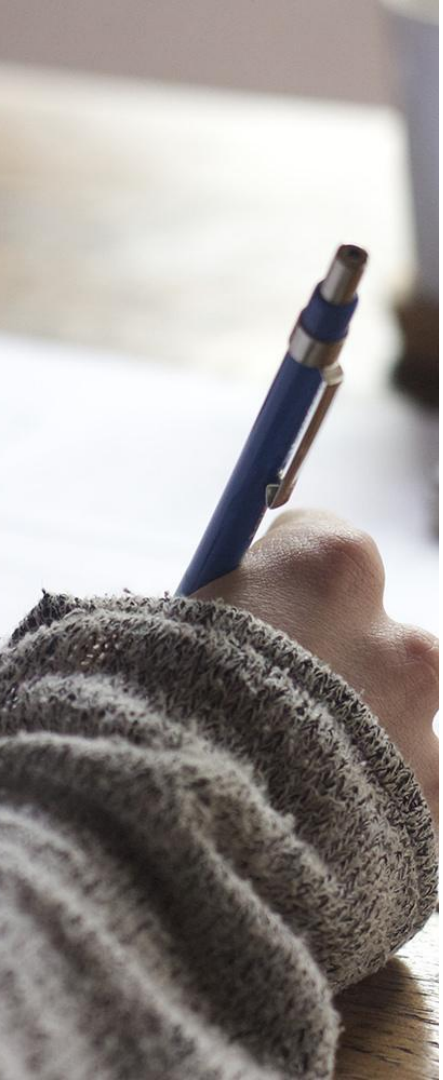
Organizing principle of data + operations

Every Relation has a Schema

- Logical Schema: describes types, names
- Physical Schema: describes data layout
- Virtual Schema (Views): derived tables

Schema:

Describes blueprint of table (s)



Set algebra (reminder)

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

A **multiset** is an unordered list (or: a set with multiple duplicate instances allowed)

UNIONs

Set: {1, 2, 3} U { 2 } = { 1, 2, 3 }
Multiset: {1, 1, 2, 3} U { 2 } = { 1, 1, 2, 2, 3 }

Cross-product

{1, 1, 2, 3} * { y, z } =
{ <1, y>, <1, y>, <2, y>, <3, y>
<1, z>, <1, z>, <2, z>, <3, z>
}

Python operating on Lists [reminder]

BASIC TYPES

Int, long int
string …

MAP + FILTER

map(function, list of inputs)

filter(function, list of inputs)

- Map applies function to input list
- Filter returns sub-list that satisfies a filter condition

REDUCE/AGGREGATE

reduce (...)

- Reduce runs a computation on a list and returns a result. E.g., SUM, MAX, MIN

For review, check out your favorite python tutorial (e.g, https://book.pythontips.com/en/latest/map_filter.html

SQL Queries on Tables (Lists of rows)

BASIC TYPES

Int32, int64
Char[n] …
Float32, float64

MAP + FILTER

Single table query

```
SELECT  c1, c2
FROM    T
WHERE  condition;
```

Multi table JOIN

```
SELECT  c1, c2
FROM    T1, T2
WHERE  condition;
```

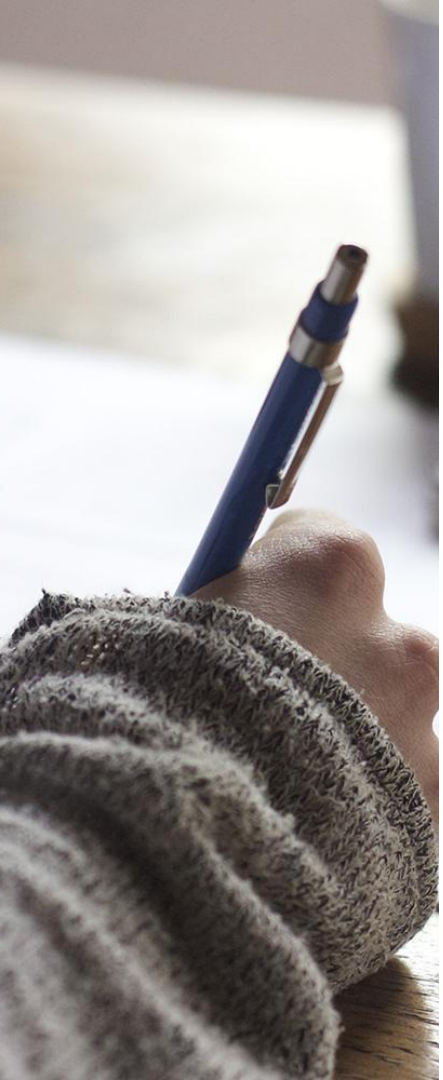
REDUCE/AGGREGATE

```
SELECT  SUM(c1*c2)
FROM    T
WHERE  condition
GROUP BY  c3;
```

Map-Filter-Reduce pattern: Same simple/powerful idea in MapReduce, Hadoop, Spark, etc.

Concepts In this section

1. SQL introduction & schema definitions
2. Basic single-table queries
3. Multi-table queries



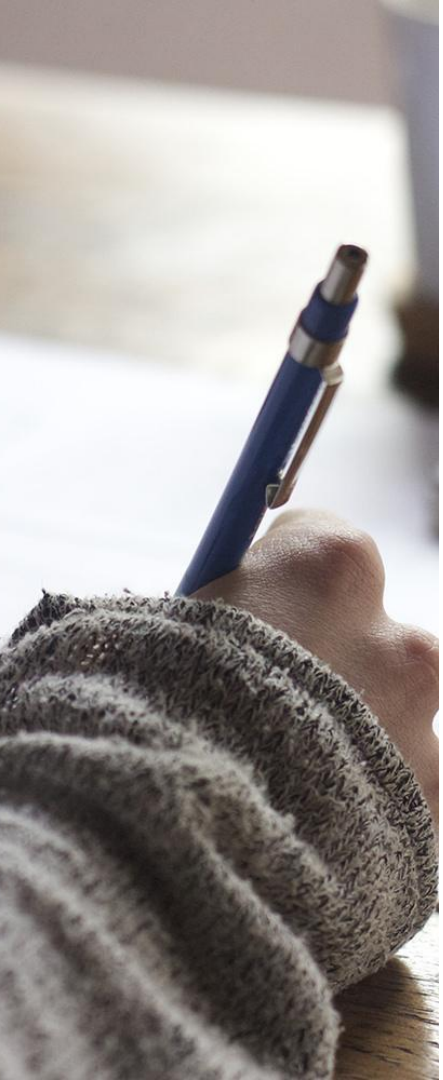
SQL Introduction

- 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), ….

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

SQL stands for
Structured
Query
Language



Data Types in SQL

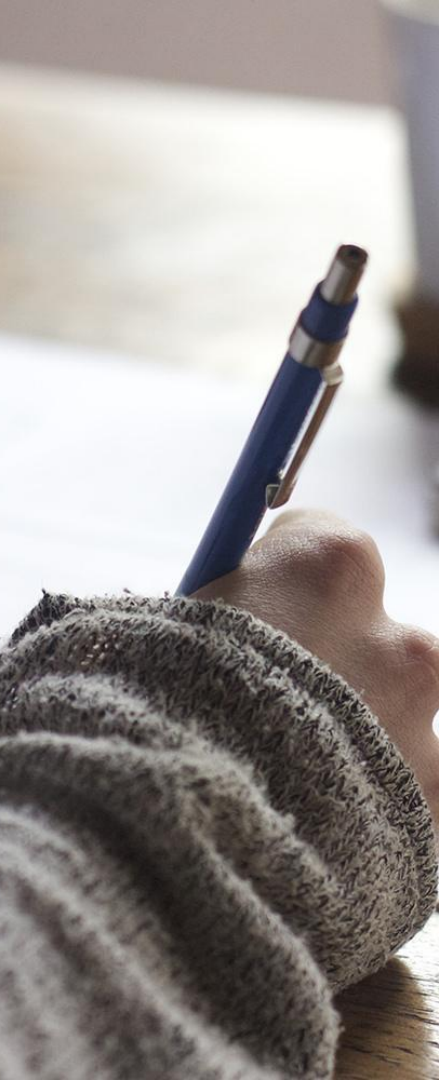
Atomic types for columns:

Characters: CHAR(20), VARCHAR(50)

Numbers: INT, BIGINT, SMALLINT, FLOAT

Others: MONEY, DATETIME…

(Most SQL dialects support **record** types, e.g. json-like. Not important for cs145.)



Tables

A **relation** or **table** is a multiset of rows, with columns.
The **schema** of a table is the table name, its columns, and their types.

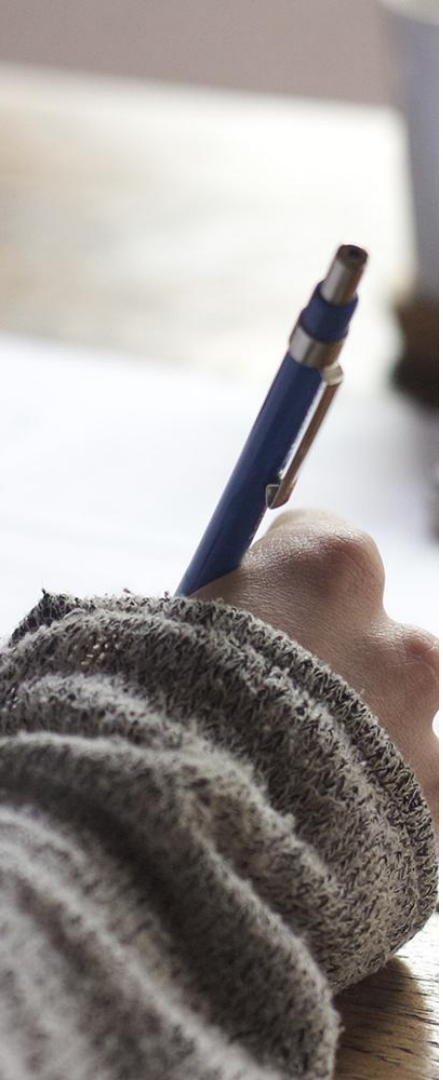
Product

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

Each **row** (or **tuple** or **record)** has attributes

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

Each **column** (or **attribute**) has a **type**
The number of columns is the **arity** of the relation.



Key constraints

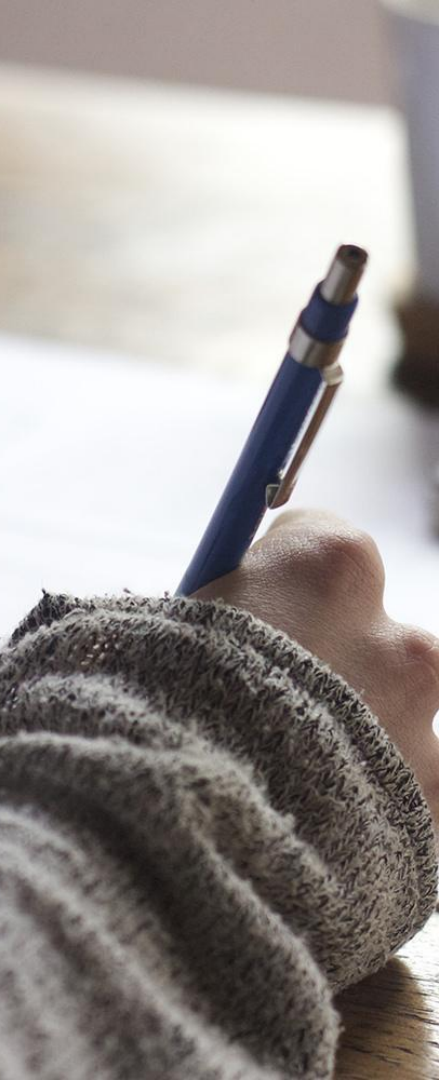
A **key** is a **minimal subset of columns** that acts as a unique identifier for tuples in a relation

- i.e. if two tuples agree on the values of the key, then they must be the same tuple!

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

Design choices?

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?



Declaring Schema

```
CREATE TABLE Product (
  Pname CHAR(20),
  Manufacturer  VARCHAR(50),
  price  float,
  Category VARCHAR(50),
  PRIMARY KEY (Pname, Manufacturer))
```

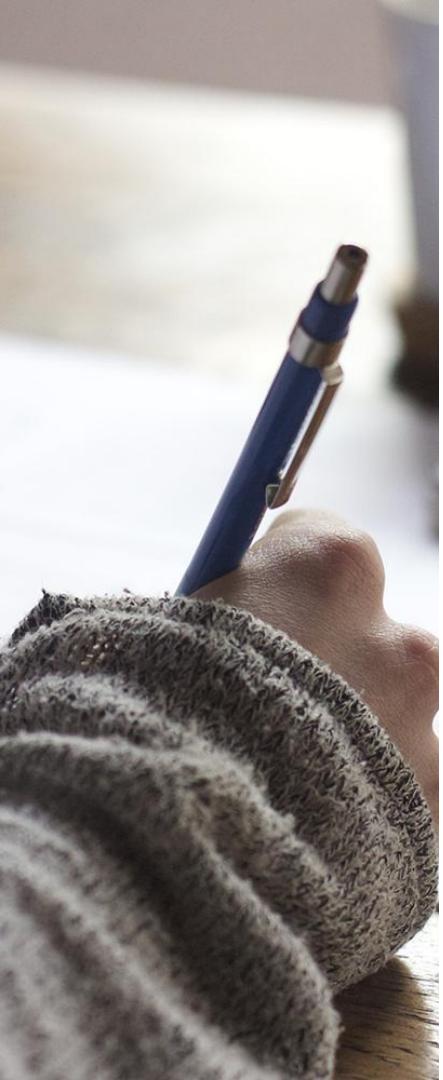
Product(Pname: *string*, Price: *float*, Category: *string*, Manufacturer: *string*)

Single - table queries

What you will learn about in this section

1. The SFW query

2. Other useful operators: LIKE, DISTINCT, ORDER BY

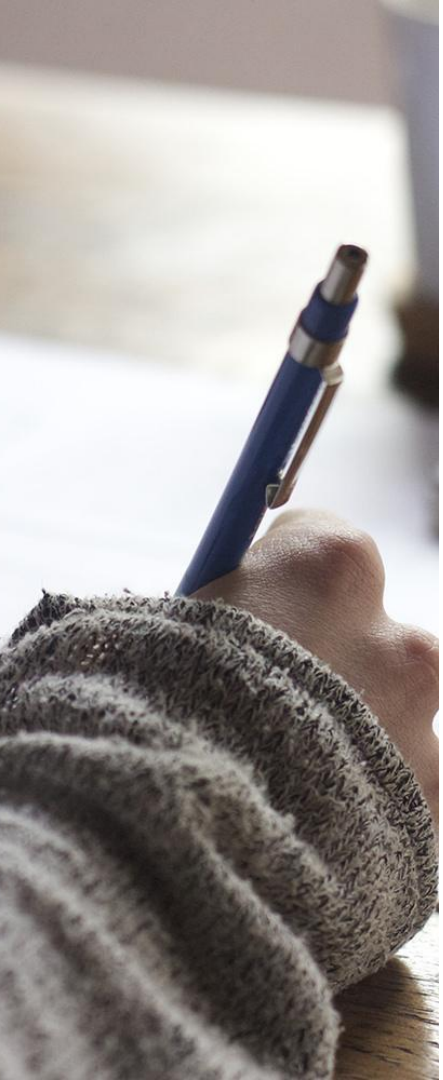


SQL Query

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

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

Call this a **SFW** query.



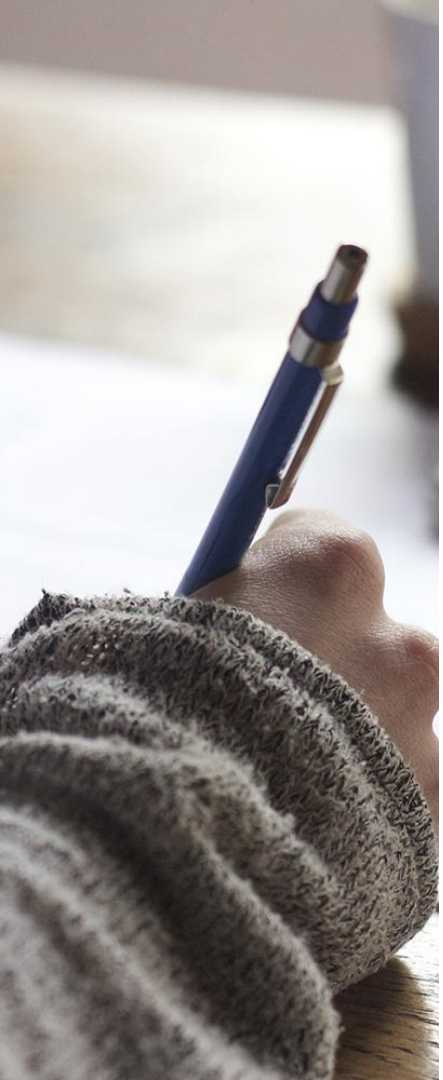
Simple SQL Query: Selection

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

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

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

PName	Price	Category	Manuf
Gizmo	$19.99	Gadgets	GWorks
Powergizmo	$29.99	Gadgets	GWorks



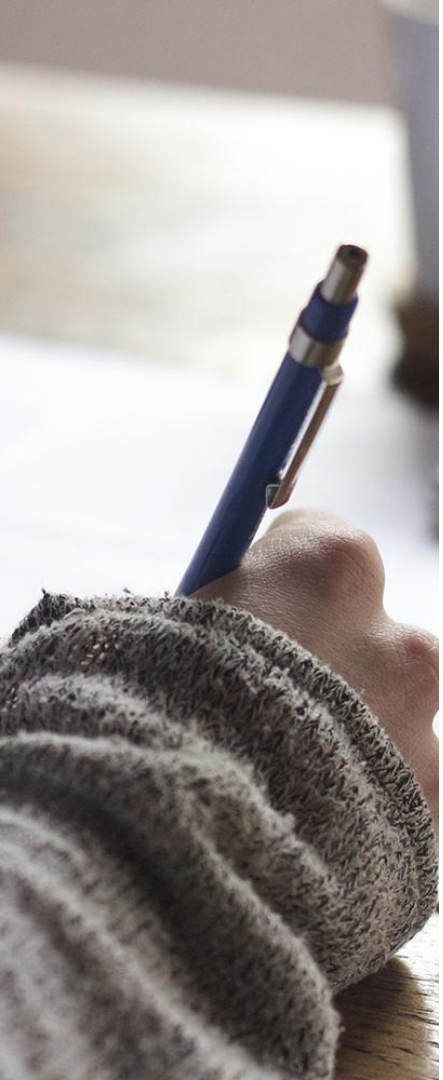
Simple SQL Query: Projection

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

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

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

PName	Price	Manuf
Gizmo	$19.99	GWorks
Powergizmo	$29.99	GWorks



Notation

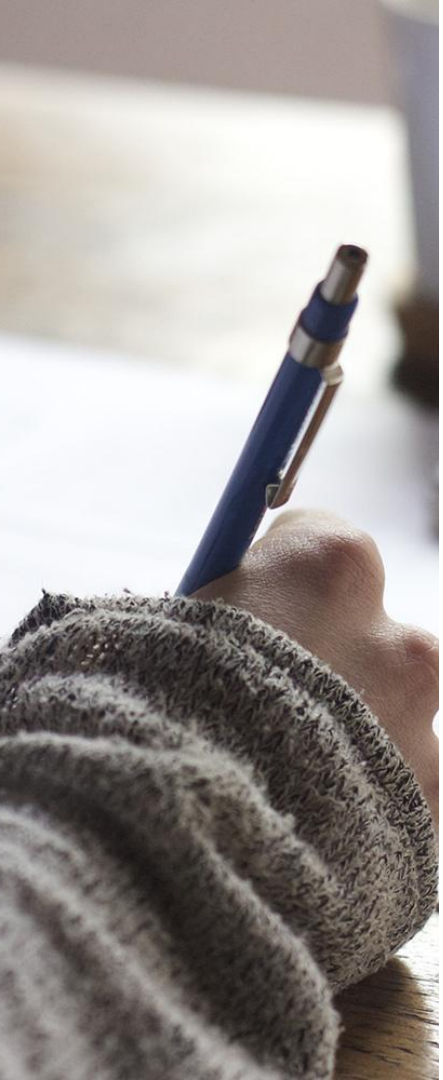
Input Schema

Product(PName, Price, Category, Manufacturer)

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

Output Schema

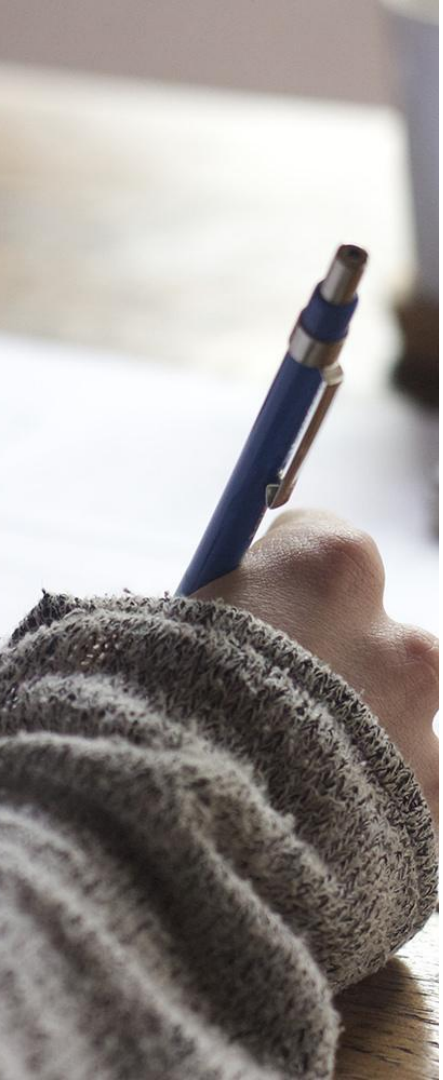
Answer(PName, Price, Manufacturer)



LIKE: Simple String Pattern Matching

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

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



DISTINCT: Eliminating Duplicates

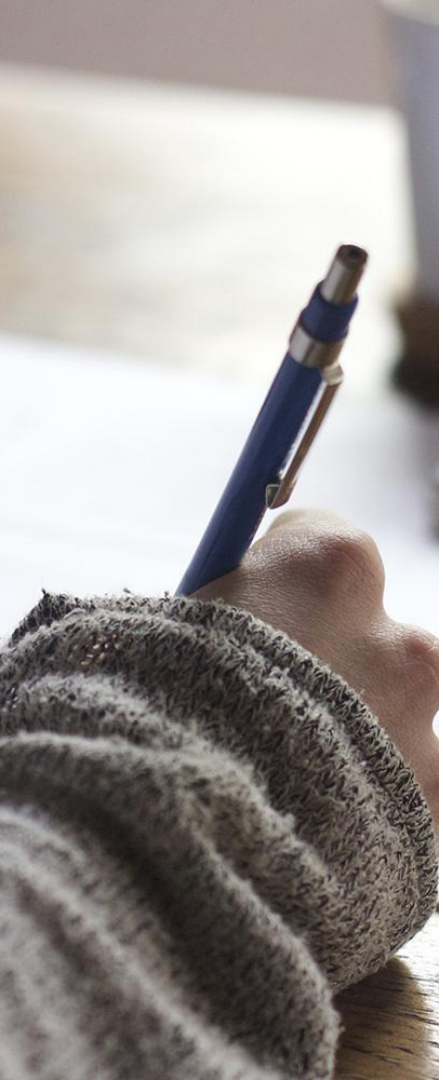
```
SELECT DISTINCT Category
FROM   Product
```

Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

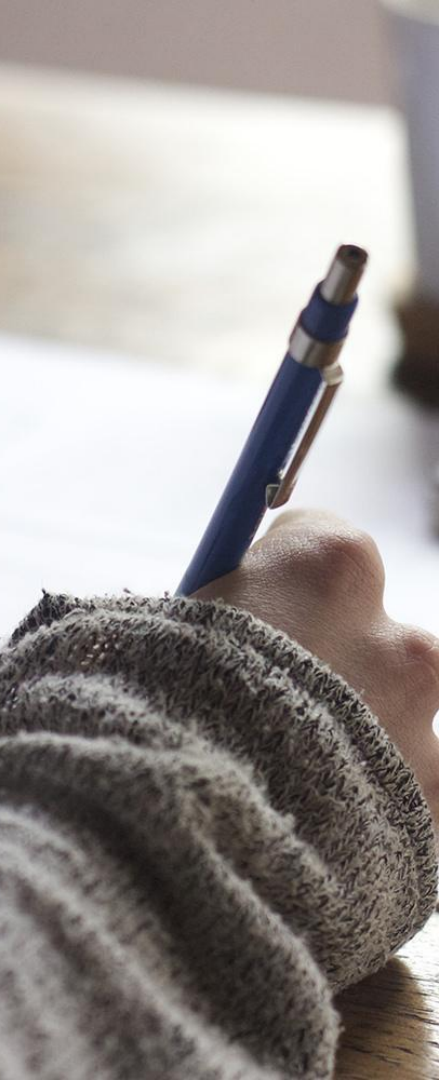


ORDER BY: Sorting the Results

```
SELECT    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.



A Few Details

1. SQL **commands** are case insensitive:
   a. SELECT = Select, Product = product

2. **Values** are **not:** ‘Seattle’ vs ‘seattle’

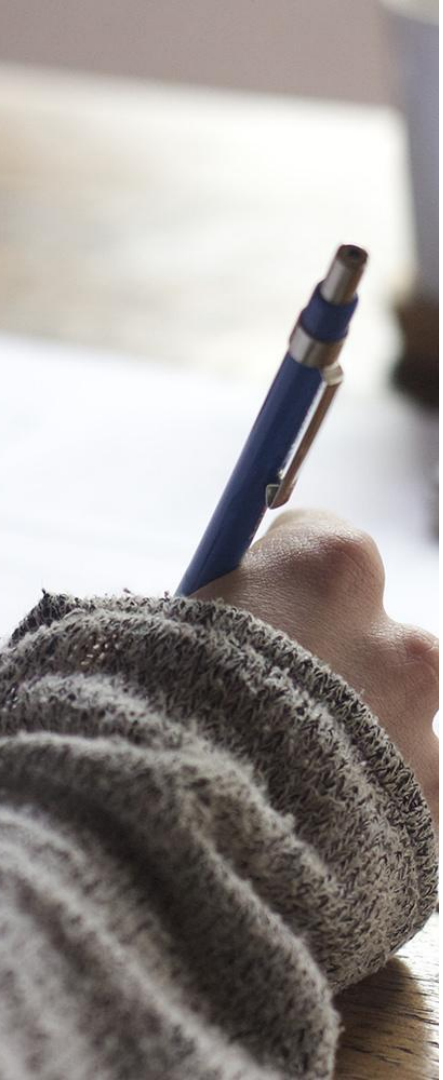
3. Use single quotes for constants:
   a. ‘abc’ - best practice (versus “abc” with mixed support)

4. To say “don’t know the value” we use NULL
   a. Common, but annoying (more detail later)
   b. E.g., Student GPA in 1st quarter = NULL, not zero

Multi-table queries

What you will learn about in this section

1. Foreign key constraints
2. Joins: basics
3. Joins: SQL semantics



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:

a student must exist in the Students table to enroll in a class

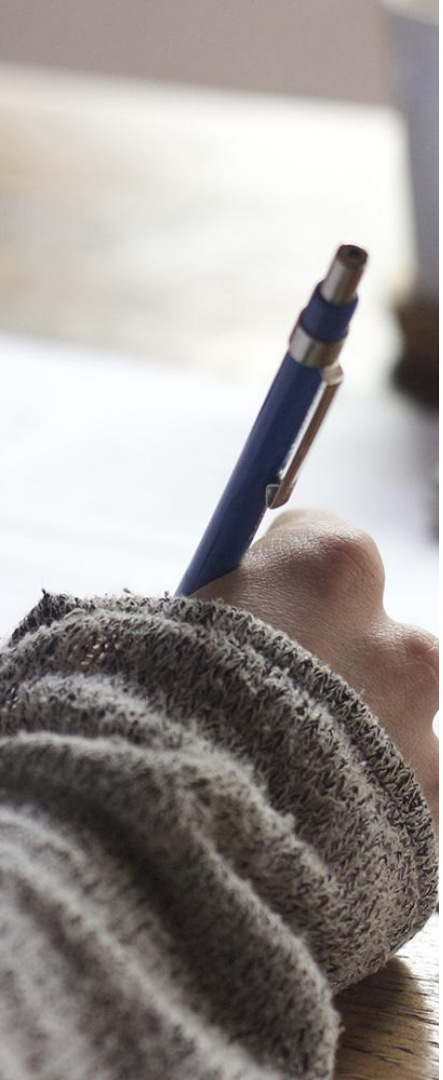
Students

sid	name	gpa
102	**Bob**	**3.9**
123	**Mary**	**3.8**

Enrolled

Student _id	cid	grade
123	564	A
123	537	A+

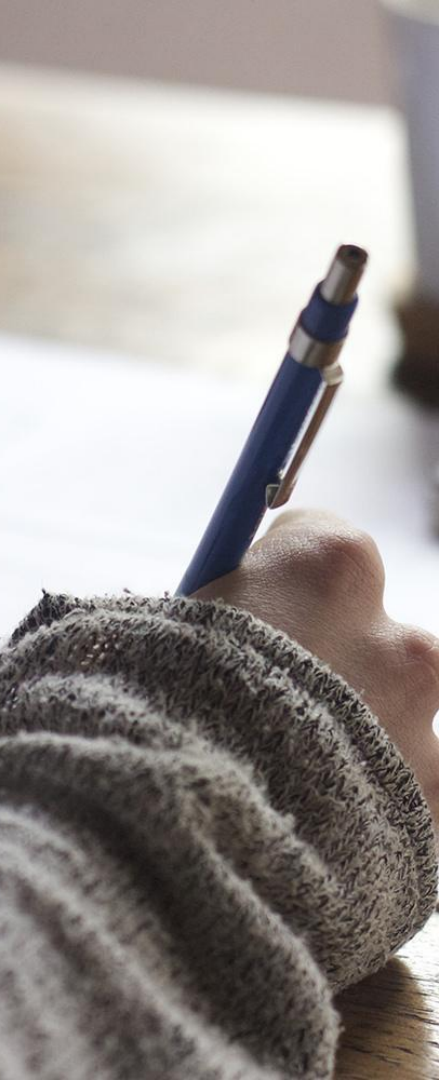
We say that student_id is a foreign key that refers to Students



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

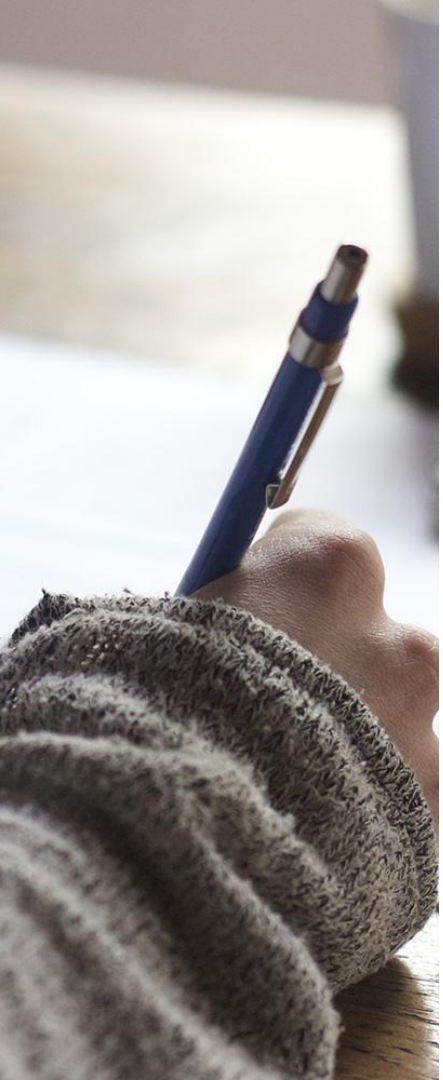
Students(sid: *string*, name: *string*, gpa: *float*)

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? Design choices
  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



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

Concepts In this section

1. SQL introduction & schema definitions
2. Basic single-table queries
3. Multi-table queries

Preview

SQL queries

sqltutorial.org/sql-cheat-sheet

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not