SQL

Part1
Introduction

Relational data model

关系数据模型

基本概念与术语

Data models 数据模型

relational data model 关系数据模型

Schemas 模式

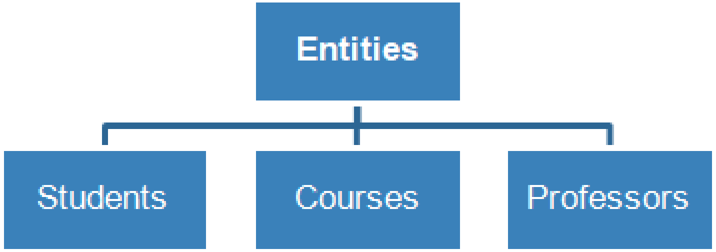
data independence 数据独立性

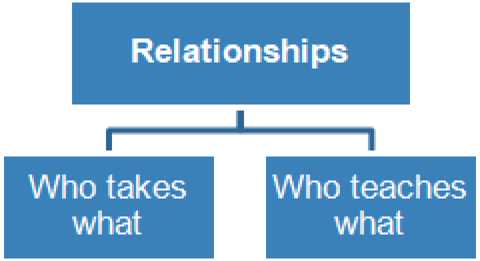
示例-课程管理系统

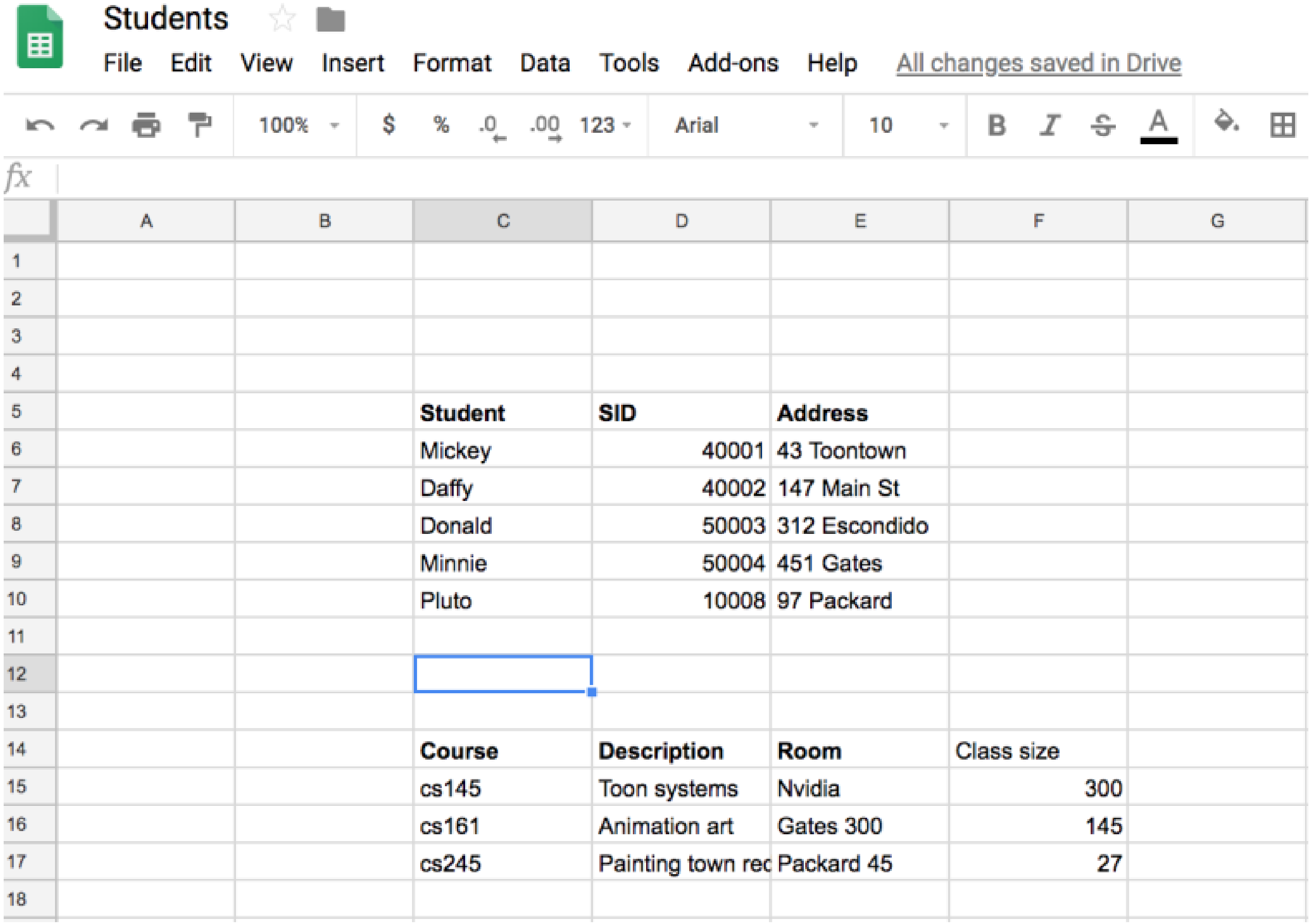
Consider building a course management system (CMS):

Entities (e.g., Students, Courses)

Relationships (e.g., Alice is enrolled in A course)







Student	SID	Address
Mickey	40001	43 Toontown
Daffy	40002	147 Main St
Donald	50003	312 Escondido
Minnie	50004	451 Gates
Pluto	10008	97 Packard

Course	Description	Room	Class size
cs145	Toon systems	Nvidia	300
cs161	Animation art	Gates 300	145
cs245	Painting town rec	Packard 45	27

‘Modeling’ the CMS

Logical Schema

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

Courses (cid: *string*, cname: *string*, credits: *int*)

Enrolled (sid: *string*, cid: *string*, grade: *string*)

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Corresponding *keys*

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

!Data model

Key concept

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)

Data independence

Key concept

Logical Data Independence

Protection from changes in the Logical Structure of the data

i.e. Should not need to ask : Can we add a new entity or attribute without rewriting the application

Physical Data Independence

Protection from Physical Layout Changes

i.e. Should not need to ask : Which disks are the data stored on? Is the data indexed?

One of the most important reasons to use a DBMS

SQL language

preview

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

Table of Contents

练习代码：SQL-1.ipynb

SQL Definitions

principles

What you will learn about in this section

1. What is SQL?

2. Basic schema definitions

3. Keys & constraints intro

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), ….

SQL stands for
Structured
Query
Language

SQL is a…

- Data Manipulation Language (DML)

  Query one or more tables

  Insert/delete/modify tuples in tables

- Data Definition Language (DDL)

  Define relational schemata

  Create/alter/delete tables and their attributes

Set algebra

List: [1, 1, 2, 3]

Set: {1, 2, 3}

<u>Multiset</u>: {1, 1, 2, 3}

UNIONs

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

<u>Multiset</u>: {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>
}

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

i.e. no *next()*, etc. methods!

Tables in SQL

Product

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

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

Also referred to sometimes as a ***Record***

Tables in SQL

Product

PName	Price	Manuf
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?

Declaring Schema

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

```
CREATE TABLE Students (
  sid CHAR(20) ,
  name VARCHAR(50) ,
  gpa float,
  PRIMARY KEY (sid),
)
```

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

多选题 1分

设置

SQL查询操作依赖的数学结构是？

A 连表List

B 集合Set

C 多集MultiSet

D 数组Array

提交

2. Single - table queries
单表查询

What you will learn about in this section

The SFW(Select-From-Where expression) query

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

<u>Selection</u> 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, Manfacturer)

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 **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
```

Ordering is ascending, unless you specify the DESC keyword.

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

3. Multiple - table queries: JOIN
多表查询：JOIN

What you will learn about in this section

JOINs

Inner JOINs

Outer JOINs

Joins

Product(<u>PName</u>, Price, Category, Manufacturer)

Company(<u>CName</u>, StockPrice, Country)

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

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
JOIN     Company
ON       Manufacturer = Cname
WHERE    Price <= 200
             AND Country='Japan'
```

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

A few more later on

Joins

Product

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

Company

CName	Stock Price	Country
GizmoWorks	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

Tuple Variable Ambiguity in Multi-Table

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

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

```
1. SELECT DISTINCT name,
   address
2. FROM Person, Company
3. 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
```

Semantics of JOINs

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

$$\begin{array}{l}\textbf{SELECT}\ \ x_1.a_1,\ x_2.a_2,\ \cdots,\ x_n.a_k \\ \textbf{FROM}\ \ R_1 \text{ AS } x_1,\ R_2 \text{ AS } x_2,\ \cdots,\ R_n \text{ AS } x_n \\ \textbf{WHERE}\ \ \text{Conditions}(x_1,\cdots,x_n)\end{array}$$

Note:

This is a *multiset* union

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

Semantics of JOINs

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

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)}

- Take **cross product**

  X = R x S

- Apply **selections/conditions**

  Y = {(r, s) in X | r.A == s.B}

- Apply **projections** to get final output

  Z = (y.A) for y in Y

= Filtering !

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on…)

An example of SQL semantics

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

R

A
1
3

S

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

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Output

A
3
3

RECAP: Joins

By default, joins in SQL are **"inner joins"**:

```
Product(name, category)
Purchase(prodName, store)
```

1

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

2

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Both equivalent:
Both INNER JOINS!

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

3

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

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Note: another equivalent way to write an INNER JOIN!

Outer Joins

- An **outer join** returns tuples from the joined relations that don't have a corresponding tuple in the other relations
    - I.e. If we join relations A and B on a.X = b.X, and there is an entry in A with X=5, but none in B with X=5…
    - A LEFT OUTER JOIN will return a tuple (a, NULL)!

- Left outer joins in SQL:

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

Now we'll get products even if they didn't sell

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

单选题 1分

设置

JOIN连接操作基于的数学运算是？

A 内积(Inner product)

B 交叉积(Cross Product)

提交

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

多选题 1分

设置

多表查询的（ JOIN ）连接操作有哪些？

A Inner JOIN

B Left JOIN

C Right JOIN

D Outer JOIN

提交

参考资料

- 斯坦福大学数据库课程
- CS145 : Data Management and Data Systems
- CS245 : Principles of Data-Intensive Systems

THANK YOU!