Introduction to Database Systems
CSE 414

Lecture 3: SQL Basics

Review

- Relational data model
  - Schema + instance + query language
- Query language: SQL
  - Create tables
  - Retrieve records from tables
  - Declare keys and foreign keys

Discussion

- Tables are NOT ordered
  - they are sets or multisets (bags)
- Tables are FLAT
  - No nested attributes

*fields/attributes cannot have tables
*cannot store tables within tables

- Tables DO NOT prescribe how they are implemented / stored on disk
  - This is called **physical data independence**

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Row major: as an array of objects

GizmoWorks USA 20000 True	Canon Japan 50000 True	Hitachi Japan 30000 True	HappyCam Canada 500 False

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Column major: as one array per attribute

GizmoWorks	Canon	Hitachi	HappyCam
USA	Japan	Japan	Canada
20000	50000	30000	500
True	True	True	False

Table Implementation

- How would you implement this?

cname	country	no_employees	for_profit
GizmoWorks	USA	20000	True
Canon	Japan	50000	True
Hitachi	Japan	30000	True
HappyCam	Canada	500	False

Physical data independence

The logical definition of the data remains unchanged, even when we make changes to the actual implementation

First Normal Form

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

- All relations must be flat: we say that the relation is in *first normal form*

First Normal Form

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

- All relations must be flat: we say that the relation is in *first normal form* (i.e. no nested elements)
- E.g., we want to add products manufactured by each company:

First Normal Form

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

- All relations must be flat: we say that the relation is in *first normal form*
- E.g., we want to add products manufactured by each company:

NOT first normal form!

cname	country	no_employees	for_profit	products
Canon	Japan	50000	Y	pname / price / category SingleTouch / 149.99 / Photography Gadget / 200 / Toy
Hitachi	Japan	30000	Y	pname / price / category AC / 300 / Appliance

First Normal Form

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

- All relations must be flat: we say that the relation is in *first normal form*
- E.g., we want to add products manufactured by each company:

Non-1NF!

cname	country	no_employees	for_profit	products
Canon	Japan	50000	Y	pname / price / category SingleTouch / 149.99 / Photography Gadget / 200 / Toy
Hitachi	Japan	30000	Y	pname / price / category AC / 300 / Appliance

First Normal Form

We will learn how different languages and data models handle this aspect

Now it's in 1NF

Company

cname	country	no_employees	for_profit
Canon	Japan	50000	Y
Hitachi	Japan	30000	Y

store elements as separate tables rather than as nested table; might require foreign key (i.e. manufacturer) to uniquely identify creator

Products

pname	price	category	manufacturer
SingleTouch	149.99	Photography	Canon
AC	300	Appliance	Hitachi
Gadget	200	Toy	Canon

SQL

- **S**tructured **Q**uery **L**anguage
- Most widely used language to query relational data
- One of the many languages for querying relational data

state what info you want, and database handles the semantics of lower-level finding/storing/sorting

- A **declarative** programming language

Selections in SQL

```
SELECT *
FROM   Product
WHERE  price > 100.0
```

WHERE keyword adds a condition; only returns tuples that satisfy the given condition.

file lec3-demo/2-demo-2.sql

Demo 2

Gives an example of defining foreign keys

*manufacturer varchar(20);
FOREIGN KEY manufacturer REFERENCES Company(cname)

this is telling the table that the attribute manufacturer sits in (Products) that the manufacturer attribute will be a foreign key, and it is a key to the primary key in Company named cname

The LIKE keyword
* Searches for substrings within attribute values (use % as you use * in terminal; wildcard)

SELECT * FROM Product WHERE category LIKE '%e%'

matches any tuple whose category value has an "e" in it

DISTINCT keyword
* SELECT DISTINCT —> removes any duplicates

ORDER BY keyword ASC/DESC
* ASC/DESC sorts with ascending or descending order
* ORDER BY manufacturer, price —> orders first alphabetically by manufacturer and uses price to break ties

Joins and joining stuff (selecting stuff from multiple tables/relations)

```
SELECT DISTINCT cname
FROM PRODUCT, Company
WHERE
country = 'USA' AND
category = 'gadget' AND
manufacturer = cname;
```

no ambiguity here as only one table
has an attribute/field with the name country
or category. Finds tuple with a specific value

LINKER; tells you what data between the
tables should be combined;
manufacturer is an attribute in one table,
and cname is attribute in the other.
So combine the data for the rows
from the different tables where
cname and manufacturer match

```
Product(pname, price, category, manufacturer)
Company(cname, country)
```

Joins in SQL

pname	price	category	manufacturer
MultiTouch	199.99	gadget	Canon
SingleTouch	49.99	photography	Canon
Gizom	50	gadget	GizmoWorks
SuperGizmo	250.00	gadget	GizmoWorks

cname	country
GizmoWorks	USA
Canon	Japan
Hitachi	Japan

Retrieve all Japanese products that cost < $150

```
Product(pname, price, category, manufacturer)
Company(cname, country)
```

Joins in SQL

pname	price	category	manufacturer
MultiTouch	199.99	gadget	Canon
SingleTouch	49.99	photography	Canon
Gizom	50	gadget	GizmoWorks
SuperGizmo	250.00	gadget	GizmoWorks

cname	country
GizmoWorks	USA
Canon	Japan
Hitachi	Japan

Retrieve all Japanese products that cost < $150

```
SELECT pname, price
FROM   Product, Company
WHERE  ...
```

```
Product(pname, price, category, manufacturer)
Company(cname, country)
```

Joins in SQL

pname	price	category	manufacturer
MultiTouch	199.99	gadget	Canon
SingleTouch	49.99	photography	Canon
Gizom	50	gadget	GizmoWorks
SuperGizmo	250.00	gadget	GizmoWorks

cname	country
GizmoWorks	USA
Canon	Japan
Hitachi	Japan

Retrieve all Japanese products that cost < $150

```
SELECT P.pname, C.price
FROM   Product as P, Company as C
WHERE  P.manufacturer=C.cname AND
       P.country='Japan' AND C.price < 150
```

Join Predicate

Selection predicates

This P. allows you to specifically tell which table the attribute is in; needed if the same attribute appears in both tables to eliminate ambiguity.

```
Product(pname, price, category, manufacturer)
Company(cname, country)
```

Joins in SQL

pname	price	category	manufacturer
MultiTouch	199.99	gadget	Canon
SingleTouch	49.99	photography	Canon
Gizom	50	gadget	GizmoWorks
SuperGizmo	250.00	gadget	GizmoWorks

cname	country
GizmoWorks	USA
Canon	Japan
Hitachi	Japan

Retrieve all USA companies that manufacture “gadget” products

Product(pname, price, category, manufacturer)
Company(cname, country)

Joins in SQL

pname	price	category	manufacturer
MultiTouch	199.99	gadget	Canon
SingleTouch	49.99	photography	Canon
Gizom	50	gadget	GizmoWorks
SuperGizmo	250.00	gadget	GizmoWorks

cname	country
GizmoWorks	USA
Canon	Japan
Hitachi	Japan

Retrieve all USA companies that manufacture “gadget” products

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

Why DISTINCT?

Demo 2 – cont'd

Joins in SQL

- The standard join in SQL is sometimes called an inner join

  cartesian product (all possible combos) of rows in the tables, and then filter using the join predicate (i.e. manufacturer = cname in last example)

  – Each row in the result **must come from both tables in the join**

  Sort of like an intersection; results are only present if they are present in BOTH tables.
  wont create any new NULL values

- Sometimes we want to include rows from only one of the two table: outer join

  more of like a union. If complementary data is not present in the other table you are joining with, will fill the unknown attributes with NULL values

```
Employee(id, name)
Sales(employeeID, productID)
```

Inner Join

Employee

id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

Retrieve employees and their sales

```
Employee(id, name)
Sales(employeeID, productID)
```

Inner Join

Employee

id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

Retrieve employees and their sales

```
SELECT *
FROM   Employee E, Sales S
WHERE  E.id = S.employeeID
```

```
Employee(id, name)
Sales(employeeID, productID)
```

Inner Join

Employee

id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

Retrieve employees and their sales

```
SELECT *
FROM   Employee E, Sales S
WHERE  E.id = S.employeeID
```

id	name	empolyeeID	productID
1	Joe	1	344
1	Joe	1	355
2	Jack	2	544

```
Employee(id, name)
Sales(employeeID, productID)
```

Inner Join

Employee

id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

She has no corresponding row in Sales!

Retrieve employees and their sales

```
SELECT *
FROM   Employee E, Sales S
WHERE  E.id = S.employeeID
```

id	name	empolyeeID	productID
1	Joe	1	344
1	Joe	1	355
2	Jack	2	544

```
Employee(id, name)
Sales(employeeID, productID)
```

Inner Join

Employee

id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

Retrieve employees and their sales

Alternative syntax

```
SELECT *
FROM   Employee E
       INNER JOIN
       Sales S
    ON E.id = S.employeeID
```

Jill is missing

id	name	empolyeeID	productID
1	Joe	1	344
1	Joe	1	355
2	Jack	2	544

```
Employee(id, name)
Sales(employeeID, productID)
```

Outer Join

Employee

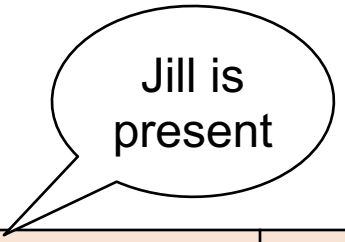
id	name
1	Joe
2	Jack
3	Jill

Sales

employeeID	productID
1	344
1	355
2	544

Retrieve employees and their sales

```
SELECT *
FROM   Employee E
       LEFT OUTER JOIN
       Sales S
    ON E.id = S.employeeID
```



id	name	empolyeeID	productID
1	Joe	1	344
1	Joe	1	355
2	Jack	2	544
3	Jill	NULL	NULL

LEFT = every row in first table (employee) will be present (i.e. if complementary data for a row in first table cannot be found in second, fill those attributes with NULL).
NOTE: if a row in second table (sales) has no corresponding table in first (employee) that row will NOT be added.
RIGHT = uses second table as its source.