Principles of Information and Database Management 198:336 Week 5 – Feb 21

Matthew Stone

Today

Joins and other advanced features of SQL

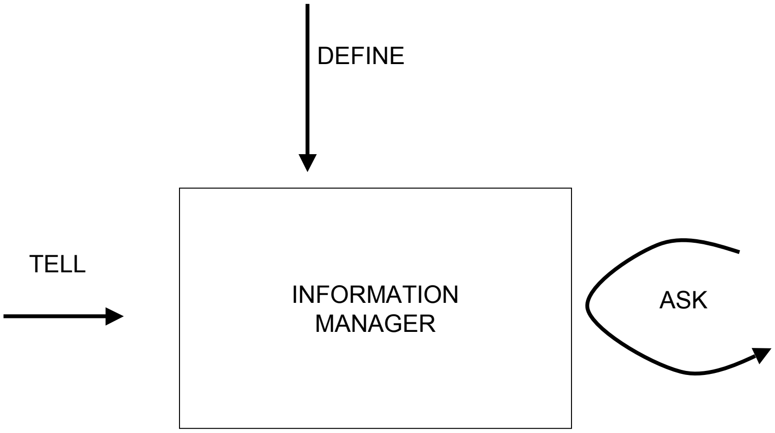
- – Recap
- – Loose ends
- – Joins
- – Optimization and relational algebra
- – Summary statistics in SQL

Recap

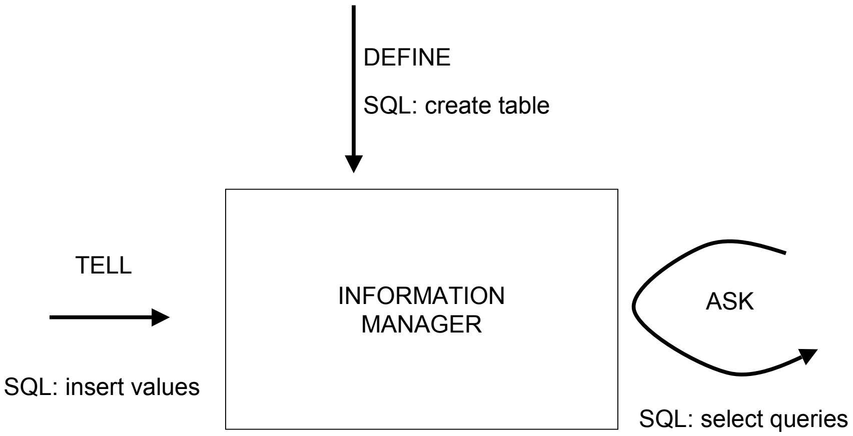
Relational model

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

In the relational model, IM stores set of relations or tables



SQL Language



Relational Model: Example

Definition statement

```
create table books
( isbn char(13) not null primary key,
  author char(80),
  title char(100),
  price float(4,2)
);
```

Relational model example

Tell statement

```
insert into books values
('0-672-31697-8',
 'Michael Morgan',
 'Java 2 for Professional Developers',
 34.99);
```

Relational model example

Tell statement

– Adds a row to the specified table in the information manager to include the specified entity or relationship.

Relational model example

Query example

```
SELECT author,title
FROM books
WHERE price > 30;
```

Relational model example

This returns a new table

author	title
Michael Morgan	Java 2 for Professional Developers
...	…

Options

not null

primary key

references table(field)

Interaction

use *database*;

show tables;

describe *table*;

Loose ends

Null values and primary keys

```
create table nullness (
  id integer primary key,
  stuff text
);

insert into nullness values
  (null, "you lose");
```

Loose ends

Updating an existing row

```
update table
   set column=expression
   where restriction
```

Update

changes columns in existing table rows

- `Set` clause indicates which columns to modify and the values they should be given.
- `Where` clause specifies which rows should be updated.

Example

Book gets “new edition”

Current row of books:

- `isbn='0-672-31697-8',`
- `author='Michael Morgan',`
- `title='Java 2 for Professional Developers',`
- `price=34.99`

Example

Command:

```
Update books
Set title = 'Java 3 for Professional Developers',
price = 39.99
where isbn='0-672-31697-8‘
```

Result: row changed to:

```
– isbn='0-672-31697-8',
– author='Michael Morgan',
– title='Java 3 for Professional Developers',
– price=39.99
```

Loose ends

Removing an existing row

`delete from` table
`where` restriction

Example

Book goes out of print

```
delete from books
where isbn='0-672-31697-8‘
```

Loose ends

Adding or deleting columns: `alter table`

Add a column, give all rows null value:

```
alter table books
add column publisher char(40)
```

Loose ends

Adding or deleting columns: `alter table`

Get rid of a column:

```
alter table books
drop column publisher
```

Loose ends

Adding or deleting columns: `alter table`

Lots of other ways to use this command.

Adding or deleting columns

Why should you not have to do this?

Loose ends

Discarding whole tables from the database

```
drop table books
```

Demo break

Any questions?

Joins - Motivation

How do you combine information from multiple tables?

Example, from book domain:

who ordered what titles?

Recap

Not useful:

```
select C.name, O.isbn
from customers C, order_items O
```

- – performs cross product on tables
- – no connections between rows

Recap

Need to establish relationships

```
select C.name, I.isbn
from customers C, orders O,
  order_items I
where C.customerid = O.customerid
and O.ordernumber = I.orderid
```

Joins

How you evaluate these queries is very important.

- Database designers describe algorithms using idea of a join – an operation that combines two tables together to give a new table.

Relational algebra

Describes operations to build relations

- Used in DB to represent query
- Can find equivalent expressions
- Can estimate how long evaluation will take

Selection

Extract rows from a relation

$$\sigma_{condition}(R)$$

extract all the rows from relation *R* that satisfy *condition*

Example

Get the rows from *S* where rating > 8

$$\sigma_{rating>8}(S)$$

Corresponds to

```
select * from S where rating > 8
```

Projection

Extract columns from a relation

$$\pi_{columns}(R)$$

make a smaller table from *R* with just the specified *columns*

Example

Extract sailor names and ratings

$$\pi_{sname,rating}(S)$$

corresponds to

```
select sname, rating from S
```

Set operations

Union $R \cup S$

Intersection $R \cap S$

Difference $R - S$

Cross-product $R \times S$

General Joins

Accepts:

- join condition
- two relations

Returns

- new relation

$$R \text{ Ä}_c\, S = \sigma_c(R \times S)$$

Equijoins

Join conditions contain only equalities

Duplicated fields are dropped

$$R \,\text{Ä}_{R.i=S.i}\, S$$

Natural join: special case

- all fields in common are equated

Equivalences

Cascading of selections

$$\sigma_{c \wedge d}(S) = \sigma_c(\sigma_d(S))$$

Commutativity of selections

$$\sigma_c(\sigma_d(S)) = \sigma_d(\sigma_c(S))$$

Equivalences

Cascading projections

$$\pi_c(\pi_d(S)) = \pi_c(S)$$

Equivalences

Commutativity of joins

$$R \otimes S = S \otimes R$$

Associativity of joins

$$R \otimes (S \otimes T) = (R \otimes S) \otimes T$$

Equivalences

Depending on the conditions at play

Selections and projections commute

$$\pi_c(\sigma_c(R)) \equiv \sigma_c(\pi_c(R))$$

Selection and join commute

$$\sigma_c(R \otimes S) \equiv \sigma_c(R) \otimes S$$

What this means in practice

The DB implementation can search for a good way to evaluate a query!