CSE 344

JULY 30TH

DB DESIGN (CH 4)
ADMINISTRIVIA

• HW6 due next Thursday
  • uses Spark API rather than MapReduce
    • (a bit higher level)
  • be sure to shut down your AWS cluster when not in use

• Still grading midterms...
  • will hand back on Wednesday
Horizontally partitioned data:

- allows for easy scale-up
  - OLTP is easy!
  - but OLAP becomes harder...
- new problem for query execution: needed data may not be local
- fix by reshuffling: put required data into the same machine
EXAMPLE 1

```
SELECT * 
FROM Order o, Line i 
WHERE o.item = i.item 
AND o.date = today()
```

Query

Single Node Plan

Multi-Node Plan
REVIEW

MapReduce

• lower-level API for
• handles networking, I/O, failure, and *straggler* issues
• main piece provided by the API is shuffle
  • read → map → shuffle → reduce → write
  • fits perfectly with what we need above
• one job contains one shuffle
  • complex queries becomes multiple jobs
PARALLEL QUERY PLANS

• Split into phases at each reshuffle

• Do as much work as possible within each phase
  • selections never require a reshuffle
  • group by & join may require it
    • (consider what data is currently partitioned by)

• In general, n-phase plan can be implemented in (n-1) MapReduce jobs
  • (since it has n-1 shuffles)
EXEMPLARY 1

Query

| SELECT * FROM Order o, Line i WHERE o.item = i.item AND o.date = today() |

Single Node Plan

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>select</td>
<td>select</td>
</tr>
<tr>
<td>date = today()</td>
<td>date = today()</td>
<td>date = today()</td>
</tr>
<tr>
<td>hash</td>
<td>hash</td>
<td>hash</td>
</tr>
<tr>
<td>o.item</td>
<td>o.item</td>
<td>o.item</td>
</tr>
<tr>
<td>h(o.item)</td>
<td>h(o.item)</td>
<td>h(o.item)</td>
</tr>
<tr>
<td>select</td>
<td>select</td>
<td>select</td>
</tr>
<tr>
<td>date = today()</td>
<td>date = today()</td>
<td>date = today()</td>
</tr>
<tr>
<td>hash</td>
<td>hash</td>
<td>hash</td>
</tr>
<tr>
<td>order o</td>
<td>order o</td>
<td>order o</td>
</tr>
<tr>
<td>scan</td>
<td>scan</td>
<td>scan</td>
</tr>
</tbody>
</table>

Multi-Node Plan

MapReduce

- saw this before in three parts: left side, right side, join
- only 1 reshuffle (for the join), so only 1 job
- mapper does first two parts: input is Item i or Order o
  - left side: output (i.item, ('Item', i))
  - right side: output (o.item, ('Order', o))
- reducer: input is (item, [(table_name, value)])
  - split list into two: items and orders
  - output all combinations (two nested loops)
EXAMPLE 2

Schema:
Drug(spec VARCHAR(255), compatibility INT)
Person(name VARCHAR(100) PK, compatibility INT)

Query
SELECT P.name, count(D.spec)
FROM Person AS P, Drug AS D
WHERE P.compatibility = D.compatibility
GROUP BY P.name;

Logical Plan
\( \gamma_{\text{name}, \text{count(spec)}} (\text{Person} \bowtie \text{Drug}) \)
EXAMPLE 2

Logical Plan

\[ \gamma_{\text{name, count(spec)}} (\text{Person} \bowtie \text{Drug}) \]

Possible Reshuffles

- join: get tuples with same “compatibility” on same machine
- group by: get tuples with same “name” on same machine
EXAMPLE 2

Logical Plan

\[ \forall \text{name, count(spec)} (\text{Person} \bowtie \text{Drug}) \]

Assume

- Drug is block partitioned
- Person is hash partitioned by “compatibility”

Physical Plan

- join: reshuffle Drug by compatibility (not needed for Person)
- group-by: no reshuffle!
  - name is a PK so only one tuple with that name
  - join can produce multiple tuples but all are on same machine with the Person tuple having that name
DB DESIGN
DATABASE DESIGN

What it is:

Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc.

Why it’s hard

The database will be in operation for a very long time (years). Updating the schema while in production is very expensive

• computer time
• engineering time
DATABASE DESIGN

Consider issues such as:
- What entities to model
- How entities are related
- What constraints exist in the domain

Several formalisms exists
- We discuss E/R diagrams
  - frequently used to communicate schemas in industry
- UML, model-driven architecture

Reading: Sec. 4.1-4.6
DATABASE DESIGN PROCESS

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details

Physical Schema
ENTITY / RELATIONSHIP DIAGRAMS

Entity set = a class
- An entity = an object

Attribute

Relationship

Product
Company
city
makes
Every entity set must have a key

- name
- price

Product
MULTIPLICITY OF E/R RELATIONS

one-one:

many-one

many-many
What does this say?
ATRIBUTES ON RELATIONSHIPS

What does this say?
MULTI-WAY RELATIONSHIPS

How do we model a purchase relationship between buyers, products and stores?

Can still model as a mathematical set (How?)

As a set of triples $\subseteq$ Person $\times$ Product $\times$ Store
Q: What does the arrow mean?

A: Any person buys a given product from at most one store.

[Fine print: Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]
Q: What does the arrow mean?

A: Any person buys a given product from at most one store AND every store sells to every person at most one product.
CONVERTING MULTI-WAY RELATIONSHIPS TO BINARY

Arrows go in which direction?
CONVERTING MULTI-WAY RELATIONSHIPS TO BINARY

Make sure you understand why!
DESIGN PRINCIPLES: WHAT'S WRONG?

Moral: Be faithful to the specifications of the application!
DESIGN PRINCIPLES: WHAT’S WRONG?

Moral: pick the right kind of entities.
DESIGN PRINCIPLES: WHAT’S WRONG?

Moral: don’t complicate life more than it already is.
FROM E/R DIAGRAMS TO RELATIONAL SCHEMA

Entity set $\rightarrow$ relation
Relationship $\rightarrow$ relation
**ENTITY SET TO RELATION**

**Product**

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokemn19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>
N-N RELATIONSHIPS TO RELATIONS

How to represent Shipment in relations?
**N-N RELATIONSHIPS TO RELATIONS**

Orders\((prod-ID, cust-ID, date)\)

Shipment\((prod-ID, cust-ID, name, date)\)

Shipping-Co\((name, address)\)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>UPS</td>
<td>4/10/2011</td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>FEDEX</td>
<td>4/9/2011</td>
</tr>
</tbody>
</table>
N-1 RELATIONSHIPS TO RELATIONS

How to represent Shipment in relations?
Orders \((prod-ID, cust-ID, date1, name, date2)\)

Shipping-Co \((name, address)\)

Remember: no separate relations for many-one relationship
MULTI-WAY RELATIONSHIPS TO RELATIONS

Product

- prod-ID
- price

Purchase

- prod-ID
- ssn
- name

Store

- name
- address

Person

- ssn
- name

Purchase(prod-ID, ssn, name)
MODELING SUBCLASSES

Some objects in a class may be special
  • define a new class
  • better: define a subclass

So --- we define subclasses in E/R
MODELING SUBCLASSES

Product

- name
- category
- price

isa

Software Product
- platforms

isa

Educational Product
- Age Group
Other ways to convert are possible...
Is this representation subclassing in Java sense?

### Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

### Sw.Product

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

### Ed.Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>toddler</td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
</tr>
</tbody>
</table>
MODELING UNION TYPES WITH SUBCLASSES

Say: each piece of furniture is owned either by a person or by a company
MODELING UNION TYPES WITH SUBCLASSES

Say: each piece of furniture is owned either by a person or by a company

Solution 1. Acceptable but imperfect (What’s wrong ?)
MODELING UNION TYPES WITH SUBCLASSES

Solution 2: better, more laborious
Entity sets are weak when their key comes from other classes to which they are related.

Team(sport, number, universityName)
University(name)
WHAT ARE THE KEYS OF R?
An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

ICs help prevent entry of incorrect information

How? DBMS enforces integrity constraints

- Allows only legal database instances (i.e., those that satisfy all constraints) to exist
- Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application
CONSTRAINTS IN E/R DIAGRAMS

Finding constraints is part of the modeling process. Commonly used constraints:

**Keys:** social security number uniquely identifies a person.

**Single-value constraints:** a person can have only one biological father.

**Referential integrity constraints:** if you work for a company, it must exist in the database.

**Other constraints:** peoples’ ages are between 0 and 120
KEYS IN E/R DIAGRAMS

No formal way to specify multiple keys in E/R diagrams
SINGLE VALUE CONSTRAINTS

makes vs. makes
Each product made by at most one company. Some products made by no company.

Each product made by exactly one company.
Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities
Constraints in SQL:
- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

The more complex the constraint, the harder it is to check and to enforce
KEY CONSTRAINTS

Product(name, category)

CREATE TABLE Product (  
    name CHAR(30) PRIMARY KEY,  
    category VARCHAR(20))

OR:

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    PRIMARY KEY (name))
KEYS WITH MULTIPLE ATTRIBUTES

Product(name, category, price)

CREATE TABLE Product ( 
    name CHAR(30), 
    category VARCHAR(20), 
    price INT, 
    PRIMARY KEY (name, category))

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
OTHER KEYS

CREATE TABLE Product (  
productID CHAR(10),  
name CHAR(30),  
category VARCHAR(20),  
price INT,  
PRIMARY KEY (productID),  
UNIQUE (name, category))

There is at most one PRIMARY KEY;  
there can be many UNIQUE
FOREIGN KEY CONSTRAINTS

CREATE TABLE Purchase (  
    prodName CHAR(30)  
    REFERENCES Product(name),  
    date DATETIME)  

prodName is a foreign key to Product(name)  
name must be a key in Product  

Referential integrity constraints  
May write just Product if name is PK
FOREIGN KEY CONSTRAINTS

Example with multi-attribute primary key

```sql
CREATE TABLE Purchase (  
    prodName CHAR(30),  
    category VARCHAR(20),  
    date DATETIME,  
    FOREIGN KEY (prodName, category)  
    REFERENCES Product(name, category)
)
```

(name, category) must be a KEY in Product
WHAT HAPPENS WHEN DATA CHANGES?

Types of updates:
In Purchase: insert/update
In Product: delete/update

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
WHAT HAPPENS WHEN DATA CHANGES?

SQL has three policies for maintaining referential integrity:

**NO ACTION** reject violating modifications (default)
**CASCADE** after delete/update do delete/update
**SET NULL** set foreign-key field to NULL
**SET DEFAULT** set foreign-key field to default value

- need to be declared with column, e.g.,
  CREATE TABLE Product (pid INT DEFAULT 42)
MAINTAINING REFERENTIAL INTEGRITY

CREATE TABLE Purchase (  
    prodName CHAR(30),  
    category VARCHAR(20),  
    date DATETIME,  
    FOREIGN KEY (prodName, category)  
        REFERENCES Product(name, category)  
    ON UPDATE CASCADE  
    ON DELETE SET NULL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Snap</td>
<td>Camera</td>
</tr>
<tr>
<td>EasyShoot</td>
<td>Camera</td>
</tr>
</tbody>
</table>
CONSTRAINTS ON ATTRIBUTES AND TUPLES

Constraints on attributes:

- **NOT NULL** -- obvious meaning...
- **CHECK** condition -- any condition!

Constraints on tuples

**CHECK** condition
CONSTRAINTS ON ATTRIBUTES AND TUPLES

```sql
CREATE TABLE R (  
  A int NOT NULL,  
  B int CHECK (B > 50 and B < 100),  
  C varchar(20),  
  D int,  
  CHECK (C >= 'd' or D > 0))
```
CONSTRAINTS ON ATTRIBUTES AND TUPLES

CREATE TABLE Product (  
    productID CHAR(10),  
    name CHAR(30),  
    category VARCHAR(20),  
    price INT CHECK (price > 0),  
    PRIMARY KEY (productID),  
    UNIQUE (name, category))
CREATE TABLE Purchase ( 
    prodName CHAR(30) 
    CHECK (prodName IN 
        (SELECT Product.name 
            FROM Product), 
        date DATETIME NOT NULL) 
What does this constraint do?

What is the difference from Foreign-Key?
GENERAL ASSERTIONS

CREATE ASSERTION myAssert CHECK
(NOT EXISTS(
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200 )

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers