Introduction to Data Management CSE 344

Lecture 17: Views

What is a View?

A view is a relation defined by a query

Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

```
CREATE VIEW StorePrice AS
SELECT x.store, y.price
FROM Purchase x, Product y
WHERE x.pid = y.pid
```

This is like a new table StorePrice(store,price)

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

How to Use a View?

- A "high end" store is a store that sold some product over 1000. For each customer, find all the high end stores that they visit. Return a set of (customer-name, high-end-store) pairs.

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.customer
AND u.store = v.store
AND v.price > 1000
```


Types of Views

- Virtual views
  - Used in databases
  - Computed only on-demand – slow at runtime
  - Always up to date
- Materialized views
  - Used in data warehouses
  - Pre-computed offline – fast at runtime
  - May have stale data
  - Indexes *are* materialized views

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

Query Modification

For each customer, find all the high end stores that they visit.

View:

```
CREATE VIEW StorePrice AS
  SELECT x.store, y.price
  FROM Purchase x, Product y
  WHERE x.product = y.pname
```

Query:

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.customer
AND u.store = v.store
AND v.price > 1000
```


Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

Query Modification

For each customer, find all the high end stores that they visit.

Modified query:

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u,
  (SELECT x.store, y.price
   FROM Purchase x, Product y
   WHERE x.pname = y.product) v
WHERE z.cid = u.customer
AND u.store = v.store
AND v.price > 1000
```

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

StorePrice(store, price)

Query Modification

For each customer, find all the high end stores that they visit.

Modified and unnested query:

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u,
     Purchase x, Product y
WHERE z.cid = u.customer
AND u.store = x.store
AND y.price > 1000
AND x.product = y.pname
```

Note that Purchase occurs twice. It has to be that way (why?).

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

AcmePurchase(cid, cname, …, price)

Further Virtual Views Optimizations

```
CREATE VIEW AcmePurchase AS
  SELECT  x.cid, x.name as cname, x.city, z.pid, z.name as pname, z.price
  FROM Customer x, Purchase y, Product z
  WHERE x.cid = y.customer and y.store = 'ACME' and y.product = z.pname
```

View

```
SELECT max(u.price)
FROM AcmePurchase u
```

Query

Find the highest prices at all ACME stores

```
SELECT  max(z.price)
FROM Customer x, Purchase y, Product z
WHERE x.cid = y.customer and y.store = 'ACME' and y.product = z.pname
```

First rewrite. Can we further optimize?

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

AcmePurchase(cid, name, …, price)

Further Virtual Views Optimizations

```
CREATE VIEW AcmePurchase AS
  SELECT  x.cid, x.name as cname, x.city, z.pid, z.name as pname, z.price
  FROM Customer x, Purchase y, Product z
  WHERE x.cid = y.cid and y.store = 'ACME' and y.pid = z.pid
```

View

```
SELECT max(u.price)
FROM AcmePurchase u
```

Query

Find the highest prices at all ACME stores

```
SELECT  max(z.price)
FROM Customer x, Purchase y, Product z
WHERE x.cid = y.customer and y.store = 'ACME' and y.product = z.pname
```

Correct if Purchase.customer is Not NULL and Foreign Key

Applications of Virtual Views

- Increased physical data independence. E.g.
  - Vertical data partitioning
  - Horizontal data partitioning
- Logical data independence. E.g.
  - Change schemas of base relations (i.e., stored tables)
- Security
  - View reveals only what the users are allowed to know

Vertical Partitioning

Resumes

SSN	Name	Address	Resume	Picture
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
234234	Ann	Portland	Clob4…	Blob4…

T1

SSN	Name	Address
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	Resume
234234	Clob1…
345345	Clob2…

T3

SSN	Picture
234234	Blob1…
345345	Blob2…

T2.SSN is a key *and* a foreign key to **T1**.SSN. Same for **T3**.SSN

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
CREATE VIEW  Resumes  AS
      SELECT  T1.ssn, T1.name, T1.address,
              T2.resume, T3.picture
      FROM    T1,T2,T3
      WHERE   T1.ssn=T2.ssn and T1.ssn=T3.ssn
```

T1(ssn,name,address)
T2(ssn,resume)
T3(ssn,picture)

Resumes(ssn,name,address,resume,picture)

Vertical Partitioning

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

```
SELECT T1.address
FROM T1, T2, T3
WHERE  T1.name = 'Sue'
 and T1.SSN=T2.SSN and T1.SSN = T3.SSN
```

Which of the tables T1, T2, T3 will be queried by the system ?

When do we use vertical partitioning ?

Vertical Partitioning Applications

1. Advantages
   - Speeds up queries that touch only a small fraction of columns
   - Single column can be compressed effectively, reducing disk I/O
2. Disadvantages
   - Updates are expensive!
   - Need many joins to access many columns
   - Repeated key columns add overhead

Hot trend today for data analytics: e.g., Vertica startup acquired by HP
They use a highly-tuned column-oriented data store AND engine

Horizontal Partitioning

Customers

SSN	Name	City
234234	Mary	Houston
345345	Sue	Seattle
345343	Joan	Seattle
234234	Ann	Portland
--	Frank	Calgary
--	Jean	Montreal

CustomersInHouston

SSN	Name	City
234234	Mary	Houston

CustomersInSeattle

SSN	Name	City
345345	Sue	Seattle
345343	Joan	Seattle

. . . . .

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
CREATE VIEW  Customers  AS
    CustomersInHouston
      UNION ALL
    CustomersInSeattle
      UNION ALL
    . . .
```


CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```

Which tables are inspected by the system ?

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```

Which tables are inspected by the system ?

All tables!
The systems doesn't know that CustomersInSeattle.city = 'Seattle'

CustomersInHouston(ssn,name,~~city~~)
CustomersInSeattle(ssn,name,~~city~~)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

Better: remove CustomerInHuston.city etc

```
CREATE VIEW  Customers  AS
    (SELECT SSN, name, 'Houston' as city
     FROM CustomersInHouston)
       UNION ALL
    (SELECT SSN, name, 'Seattle' as city
     FROM CustomersInSeattle)
       UNION ALL
    . . .
```


CustomersInHouston(ssn,name)
CustomersInSeattle(ssn,name)
. . . . .

Customers(ssn,name,city)

Horizontal Partitioning

```
SELECT  name
FROM    Customers
WHERE  city = 'Seattle'
```

⇩

```
SELECT name
FROM   CustomersInSeattle
```


Horizontal Partitioning Applications

- Performance optimization
  - Especially for data warehousing
  - E.g. one partition per month
  - E.g. archived applications and active applications
- Distributed and parallel databases
- Data integration

Levels of Abstraction