

Introduction to Database Systems

CSE 444

Lecture 4: Views and Constraints

Outline

- **Views**: Sections 8.1, 8.2, 8.3
 - [Old edition, Sections 6.6 and 6.7]
- **Constraints**: Sections 2.3, 7.1, 7.2
 - [Old edition: Sections 7.1 and 7.2 only]
- Won't discuss updates ! In sections...

Views

Views are relations, except that they may not be physically stored

For presenting different information to different users

`Employee(ssn, name, department, project, salary)`

```
CREATE VIEW Developers AS
  SELECT name, project
  FROM Employee
  WHERE department = 'Development'
```

Payroll has access to `Employee`, others only to `Developers`

Example

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

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

CustomerPrice(customer, price)

“virtual table”

Example

Purchase(customer, product, store)

Product(pname, price)

CustomerPrice(customer, price)


We can later use the view just like any other relation :

```
SELECT DISTINCT u.customer, v.store
FROM CustomerPrice u, Purchase v
WHERE u.customer = v.customer AND
      u.price > 100
```

Types of Views

- Virtual views

- Used in databases
- Computed only on-demand – slow at runtime
- Always up to date



We discuss
only virtual
views in class

- Materialized views

- Used in data warehouses
- Pre-computed offline – fast at runtime
- May have stale data
- Indexes *are* materialized views (read book)

Queries Over Views: Query Modification

View:

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

Query:

```
SELECT DISTINCT u.customer, v.store
FROM CustomerPrice u, Purchase v
WHERE u.customer = v.customer AND
      u.price > 100
```

Queries Over Views: Query Modification

Modified query:

```
SELECT DISTINCT u.customer, v.store
FROM (SELECT x.customer, y.price
      FROM Purchase x, Product y
      WHERE x.product = y.pname) u, Purchase v
WHERE u.customer = v.customer AND
      u.price > 100
```


Queries Over Views: Query Modification

Modified and unnested query:

```
SELECT DISTINCT x.customer, v.store
FROM Purchase x, Product y, Purchase v,
WHERE x.customer = v.customer AND
      y.price > 100 AND
      x.product = y.pname
```

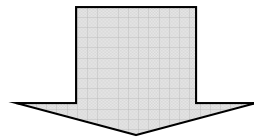
Applications of Virtual Views

- **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...

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 T2.ssn=T3.ssn
```

Vertical Partitioning

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

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

When do we use vertical partitioning ?

Vertical Partitioning Applications

1. Can improve performance of some queries
 - When queries touch small fraction of columns
 - Only need to read desired columns from disk
 - Can produce big I/O savings for wide tables
 - Potential benefit in data warehousing applications
- But
 - Repeated key columns add a lot of overhead
 - Need expensive joins to reconstruct tuples

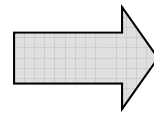
Vertical Partitioning Applications

2. When some fields are large and rarely accessed
 - E.g. Picture
3. In distributed databases
 - Customer personal info at one site, profile at another
4. In data integration
 - T1 comes from one source
 - T2 comes from a different source

Horizontal Partitioning

Customers

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



CustomersInHouston

SSN	Name	City	Country
234234	Mary	Houston	USA

CustomersInSeattle

SSN	Name	City	Country
345345	Sue	Seattle	USA
345343	Joan	Seattle	USA

CustomersInCanada

SSN	Name	City	Country
--	Frank	Calgary	Canada
--	Jean	Montreal	Canada

Horizontal Partitioning

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

Horizontal Partitioning

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

Which tables are inspected by the system ?

WHY ???

Horizontal Partitioning

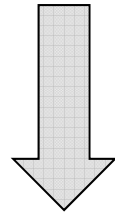
Better:

```
CREATE VIEW Customers AS
  (SELECT * FROM CustomersInHouston
   WHERE city = 'Houston')
  UNION ALL
  (SELECT * FROM CustomersInSeattle
   WHERE city = 'Seattle')
  UNION ALL
  . . .
```

Other techniques exist: read DBMS documentation

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

Views and Security

Customers:

Name	Address	Balance
Mary	Houston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

Fred is not allowed to see this

Fred is allowed to see this

```
CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers
```

Views and Security

Customers:

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

Wilma is
not allowed
to see >0
balances

```
CREATE VIEW BadCreditCustomers
SELECT *
FROM Customers
WHERE Balance < 0
```

Outline

- **Views**: Sections 8.1, 8.2, 8.3
 - [Old edition, Sections 6.6 and 6.7]
- **Constraints**: Sections 2.3, 7.1, 7.2
 - [Old edition: Sections 7.1 and 7.2 only]

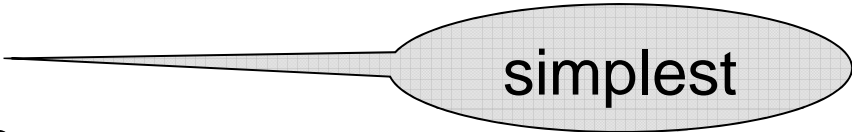
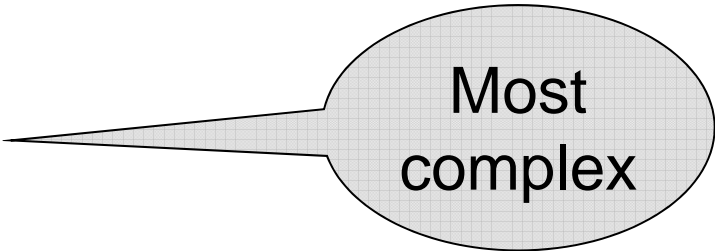
Integrity Constraints Motivation

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
- 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

Types of Constraints in SQL

Constraints in SQL:

- Keys, foreign keys  simplest
 - Attribute-level constraints
 - Tuple-level constraints
 - Global constraints: assertions  Most complex
- 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))
```

Name	Category	Price
Gizmo	Gadget	10
Camera	Photo	20
Gizmo	Photo	30
Gizmo	Gadget	40

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)
```

Referential
integrity
constraints

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

May write
just Product
(why ?)

Foreign Key Constraints

Product

<u>Name</u>	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz



Foreign Key Constraints

- Example with multi-attribute primary key

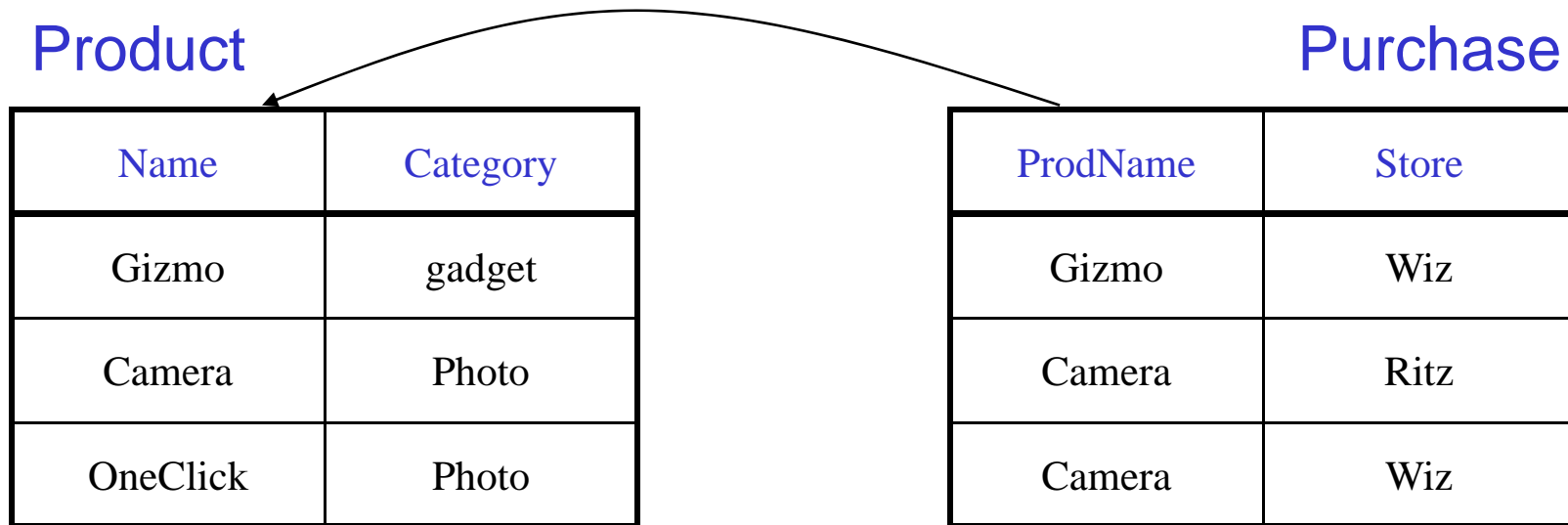
```
CREATE TABLE Purchase (  
    prodName CHAR(30),  
    category VARCHAR(20),  
    date DATETIME,  
    FOREIGN KEY (prodName, category)  
    REFERENCES Product(name, category)
```

- (name, category) must be a PRIMARY KEY in Product

What happens during updates ?

Types of updates:

- In Purchase: insert/update
- In Product: delete/update



What happens during updates ?

- SQL has three policies for maintaining referential integrity:
- Reject violating modifications (default)
- Cascade: after delete/update do delete/update
- Set-null set foreign-key field to NULL

READING ASSIGNMENT: 7.1.2 and 7.1.3
[Old edition: 7.1.5, 7.1.6]

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

What
is the difference from
Foreign-Key ?

```
CREATE TABLE Purchase (  
  prodName CHAR(30)  
  CHECK (prodName IN  
    (SELECT Product.name  
     FROM Product),  
  date DATETIME NOT NULL)
```

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
Instead, they provide triggers
To learn more, read the rest of Chapter 7