Views
Views are relations, except that they are not physically stored.
For presenting different information to different users

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

Payroll has access to Employee, others only to Developers

Example

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

CustomerPrice(customer, price) “virtual table”

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

Modified query:
SELECT 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

Modified and rewritten query:
SELECT 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

But What About This ?

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

??

Applications of Virtual Views

• Logical data independence:
  – Vertical data partitioning
  – Horizontal data partitioning

• Security
  – Table (view) V reveals only what the users are allowed to know

Answer

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

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
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

When do we use vertical partitioning?

Vertical Partitioning

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

Vertical Partitioning

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

Horizontal Partitioning

CREATE VIEW Customers AS
CustomersInHouston
UNION ALL
CustomersInSeattle
UNION ALL
CustomersInCanada

Horizontal Partitioning

Customers

Vertical Partitioning

Resumes

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary Huston</td>
<td>Clob1... Blob1...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>345345</td>
<td>Sue Seattle</td>
<td>Clob2... Blob2...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>345343</td>
<td>Joan Seattle</td>
<td>Clob3... Blob3...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>234234</td>
<td>Ann Portland</td>
<td>Clob4... Blob4...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary Huston</td>
<td></td>
</tr>
<tr>
<td>345345</td>
<td>Sue Seattle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Clob1...</td>
</tr>
<tr>
<td>345345</td>
<td>Clob2...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Blob1...</td>
</tr>
<tr>
<td>345345</td>
<td>Blob2...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Horizontal Partitioning

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

Which tables are inspected by the system?

WHY ???

Better:

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

Horizontal Partitioning

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

Applications:

- Optimizations:
  - E.g. archived applications and active applications
- Distributed databases
- Data integration

Views and Security

<table>
<thead>
<tr>
<th>Customers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Sue</td>
</tr>
<tr>
<td>Joan</td>
</tr>
<tr>
<td>Ann</td>
</tr>
</tbody>
</table>

Fred is not allowed to see this

CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers

Fred is allowed to see this

CREATE VIEW BadCreditCustomers
SELECT * FROM Customers
WHERE Balance < 0

John is allowed to see only <0 balances

Views and Security

<table>
<thead>
<tr>
<th>Customers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Sue</td>
</tr>
<tr>
<td>Joan</td>
</tr>
<tr>
<td>Ann</td>
</tr>
</tbody>
</table>
Constraints in SQL

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.

Keys

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

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>
</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.

Name   | Category | Store
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
Foreign Key Constraints

- OR

```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 PRIMARY KEY

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 a delete/update do a delete/update
  - Set-null set foreign-key field to NULL

READING ASSIGNMENT: 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

General Assertions

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