Part I

Views and Access Control
Views and Access Control

1 View Concept
Views and Access Control

1 View Concept

2 Changes via View
Views and Access Control

1. View Concept
2. Changes via View
3. Assignment of Rights
Views and Access Control

1. View Concept
2. Changes via View
3. Assignment of Rights
4. Privacy-Aspects
Learning goals for today . . .

- Understanding of the view concept of databases
Learning goals for today . . .

- Understanding of the view concept of databases
- Knowledge to formalize and to use views in SQL
- Knowledge of the problems with changes via views
- Knowledge of data protection aspects in context with aggregated/statistic data
Learning goals for today . . .

- Understanding of the view concept of databases
- Knowledge to formalize and to use views in SQL
- Knowledge of the problems with changes via views
Learning goals for today . . .

- Understanding of the view concept of databases
- Knowledge to formalize and to use views in SQL
- Knowledge of the problems with changes via views
- Knowledge to data protection aspects in context with aggregated/statistic data
Views

Views: virtual relations (resp. virtual database objects in other data models)

- View are external DB-schemes that follow the 3-level-scheme architecture
- View definition
  - Relation scheme (implicit or explicit)
  - Calculation role for virtual relations, such as SQL-query
Views /2

● Advantages
  ▶ Simplification of queries for the user of the database, e.g. by realization of often required sub-queries
  ▶ Possibility of structuring of the database description, specific to user classes
  ▶ Logic data independence enables robustness of the interface for applications against changes to the database structure (accordingly vice verse)
  ▶ Description of accesses on the database in context with the access control

● Problems
  ▶ Automatic query transformation
  ▶ Execution of changes on views
Three-Level-Scheme-Architecture
Definition of Views in SQL

```sql
create view ViewName [ SchemeDeclaration ]
as SQLQuery
[ with check option ]
```
Views - Example

- all red wines from Bordeaux:

```sql
create view RedWines as
    select Name, Vintage, WINES.Vineyard
    from WINES natural join PRODUCER
    where Color = 'Red'
    and Region = 'Bordeaux'
```
Problem Areas of Views

- Execution of changes via views
- Automatic query transformation
Criterion for Changes via Views

- **Effect Conformity**
  User sees effect as if the change was done directly on the view relation.

- **Minimality**
  Basis database should only be changed minimal to preserve the mentioned effect.

- **Consistency Preservation**
  Changes of a view must not lead to integrity violations of the basis database.

- **Respecting the Database Protection**
  If a view is implemented for data protection purposes, then the consciously fade out part of the basis database must not be effected by changes of the view.
Projection View

$\text{WNW} := \pi_{\text{WineID, Name, Vineyard}}(\text{WINES})$

- In SQL with `create view`-statement:

  ```sql
  create view WNW as
  select WineID, Name, Vineyard from WINES
  ```

- Change statement for the view `WNW`:

  ```sql
  insert into WNW values (3333, 'Dornfelder', 'Müller')
  ```

- Corresponding statement on the basis relation `WINES`:

  ```sql
  insert into WINES
  values (3333, 'Dornfelder', null, null, null, 'Müller')
  ```

→ Problem of Consistence preservation if Color or Vintage declared as `not null`!
Selection Views

\[ WJ := \sigma_{\text{Vintage} > 2000}(\pi_{\text{WineID}, \text{Vintage}}(\text{WINES})) \]

create view WJ as
select WineID, Vintage
from WINES
where Vintage > 2000

Tuple migration: Tuple
\( \text{WINES}(3456, 'Zinfandel', 'Red', 2004, 'Helena') \), gets "moved out" of the view:

update WINES
set Vintage = 1998
where WineID = 3456
Kontrol of Tuple Migration

```sql
create view WJ as
select WineID, Vintage
from WINES
where Vintage > 2000
with check option
```
Join Views

\[ WE := \text{WINES} \bowtie \text{PRODUCER} \]

- In SQL:

```sql
create view WE as
select WineID, Name, Color, Vintage, WINES.Vinyard, Area, Region
from WINES, PRODUCER
where WEINE.Vineyard = ERZEUGER.Vineyard
```

- Change options usually not clearly translatable:

```sql
insert into WE
values (3333, 'Dornfelder', 'Red', 2002, 'Helena', 'Barossa Valley', 'South Australia')
```
Join Views /2

- Change is transformed to

```sql
insert into WINES
values (3333, 'Dornfelder', 'Red', 2002, 'Helena')
```

- Plus

  1. Insert statement on ERZEUGER:

```sql
insert into PRODUCER
values ('Helena', 'Barossa Valley', 'South Australia')
```

  2. Or alternative:

```sql
update PRODUCER
set Area = 'Barossa Valley', Region = 'South Australia'
where Vineyard = 'Helena'
```

better regarding **minimality requirement**, but contradicts effect conformity!
Aggregation Views

```
create view FM (Color, MinVintage) as
select Color, min(Vintage)
from WINES
group by Color
```

- Following changes are not clearly realizable:

```
update FM
set MinVintage = 1993
where Color = 'Red'
```
Classification of Problem Areas

1. Violation of the scheme definition (e.g. introduction of null values at projection view)
2. Data protection: Avoid side effects on invisible part of the database (tuple migration, selection views)
3. Not always clear transformation: choice problem
4. Aggregation views (among others): no useful transformation possible
5. Elemental view changes should exactly comply with a atomic change on basis relation: 1:1-Relation between view tuples and tuples of the basis relation (no projection of keys)
Handling of Views in SQL

- SQL-92-Standard
  - Integrity violating view changes are prohibited
  - Data protection violating view changes: user control (with check option)
  - View with unclear transformation: view not changeable (SQL-92 more restrictive than necessary)
Restrictions for View Changes

- Only selection and projection views changeable (join and set operations prohibited)
- 1:1-Relation of view tuples to basis tuples: no `distinct` in projection view
- Arithmetic and aggregation functions in the `select`-part are prohibited
- Exactly one reference on one relation name in the `from`-part permitted (also no self join)
- No sub-queries with "self reference" in the `where`-part permitted (use relation name in the top SFW-block not in the `from`-parts of sub-queries)
- `group by` and `having` prohibited
Evaluation of Queries on Views

- **select**: View attributes probably rename resp. replace by calculation term
- **from**: Names of the original relations
- Conjunctive linking of the **where**-clauses of the view definition and queries (probably renaming)
Problems with Aggregation Views

```
create view FM (Color, MinVintage) as
select Color, min(Vintage)
from WINES
group by Color
```

Query: *Wine colors with old vintages*

```
select Color
from FM
where MinVintage < 1995
```
Problems with Aggregation Views /2

- After syntactic transformation:

```sql
select Color
from WINES
where min(Vintage) < 1995
group by Color
```

- No syntactic correct SQL-query – correct would be:

```sql
select Color
from WINES
group by Color
having min(Vintage) < 1995
```
Problems with Aggregation Views /3

- **Query**

```sql
select max (MinVintage) 
from FM
```

- Should be transformed as follows:

```sql
select max(min (Vintage)) 
from WINES 
group by Color
```

- **But:** Nested aggregation functions are prohibited in SQL!
Assignment of Rights in Databases

- **Access rights**
  
  (AuthorizationID, DB-Excerpt, Operation)

- AuthorizationID is internal identification of a "database user"
- Database excerpts: relations and views
- DB-Operations: read, insert, change, remove
Assignment of Rights in SQL

grant <Rights>
on <Table>
to <UserList>
[with grant option]
Assignment of Rights in SQL /2

- Explanations:
  - In `<Rights>`-List: **all** resp. long form **all privileges** or list of `select`, `insert`, `update`, `delete`
  - After **on**: relation and view name
  - After **to**: Authorization identifications (also **public, group**)
  - Special right: right on passing of rights (**with grant option**)
Authorization for **public**

```sql
create view MyJobs as
select *
from JOB
where KName = user;

grant select, insert
on MyJobs
to public;
```

"Every user can see her jobs and can insert new jobs (but not remove!)."
Taking Back of Rights

```
revoke <Rights>
on <Table>
from <UserList>
[restrict | cascade ]
```

- **restrict**: If rights already passed to thirds: abort of *revoke*
- **cascade**: Propagate revocation of the rights with *revoke* to all users that received them from this user with *grant*
Privacy: Term and Areas of Application

Privacy: The right of each individual on a save and private room, that can only be violated by others in exceptional cases.

- Electronic highway toll system: Monitoring of vehicles
- Credit card activities and diverse payback resp. discount cards: buying behavior of customers
- Mobile communication systems: movement profiles of users
- RFID-technology: e.g. in retail trade the customer behavior, flow of goods, etc.
Statistic Databases

- Databases in which single entries are subject to data protection, but statistic information about all users is accessible
- Statistic information = aggregated data (average income etc.)
- Problem: Extraction of single information with indirect queries
Statistic Databases: Example

Example: User X can query data about the account holder as well as statistic data, but no single account balances

1. Simplification of search criterion (only one customer gets selected)

```
select count(*) from ACCOUNT
where Place = 'Manebach' and Age = 24 and ...
```

2. Name of the account holder

```
select Name from ACCOUNT
where Place = 'Manebach' and Age = 24 and ...
```

3. Statistic query, that actually gives a single entry

```
select sum(Balance) from ACCOUNT
where Place = 'Manebach' and Age = 24 and ...
```

Remedy: no query that select less than \( n \) tuples
Statistic Database: Example \(\text{/2}\)

- \(X\) wants to find out balance of \(Y\)
- \(X\) knows, that \(Y\) does not live in Ilmenau
- \(X\) has queried, that more than \(n\) account holders live in Ilmenau
  
  1. Sum of the balances of customers from Ilmenau
     
     \[
     \text{select sum(Balance) from Account}
     \text{where Place = 'Ilmenau'}
     \]
  
  2. Sum of the balances of customers from Ilmenau + Customer \(Y\)
     
     \[
     \text{select sum(Balance) from Account}
     \text{where Name = :Y or Place = 'Ilmenau'}
     \]
  
  3. Difference of the results gives balance of \(Y\)

- Remedy: prohibition of statistic queries that affect pairwise an average of more than \(m\) given tuples
Statistic Databases: Conclusion

- Critical parameters
  - Result size $n$
  - Size of the overlapping of the result set $m$

If only results of aggregate functions are permitted, than a person needs $1 + (n - 2)/m$ queries to determine a single attribute value.
k-Anonymity

- For many purposes (clinical studies etc.) detail data (micro data) is required

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>ZIP</th>
<th>Gender</th>
<th>MaritalState</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>*****</td>
<td>38</td>
<td>98693</td>
<td>male</td>
<td>married</td>
<td>cold</td>
</tr>
<tr>
<td>*****</td>
<td>29</td>
<td>39114</td>
<td>female</td>
<td>single</td>
<td>fever</td>
</tr>
<tr>
<td>*****</td>
<td>29</td>
<td>39114</td>
<td>female</td>
<td>single</td>
<td>anemia</td>
</tr>
<tr>
<td>*****</td>
<td>34</td>
<td>98693</td>
<td>male</td>
<td>married</td>
<td>cough</td>
</tr>
<tr>
<td>*****</td>
<td>34</td>
<td>98693</td>
<td>male</td>
<td>married</td>
<td>broken bone</td>
</tr>
<tr>
<td>*****</td>
<td>27</td>
<td>18055</td>
<td>male</td>
<td>single</td>
<td>fever</td>
</tr>
<tr>
<td>*****</td>
<td>27</td>
<td>18055</td>
<td>female</td>
<td>single</td>
<td>cold</td>
</tr>
</tbody>
</table>
k-Anonymity: Problem

- Is for a person of this relation known that he is:
  - male
  - 38 years old
  - married
  - living in 98693 Ilmenau

- ~≈ cold

- Further relation (Name etc.), e.g. by join with other data

- Solution: Data Swapping (??)
**k-Anonymity**

**k-Anonymity**: a certain fact cannot be differentiated among a given amount of $k$ tuples

- A query for an arbitrary combination of age, gender, marital state and ZIP code gives either an empty relation or at least $k$ tuples
k-Anonymity: Approaches

- **Generalization**: Replace attribute values by more general values that are gathered from a generalization hierarchy
  - Generalization of the age of the person to age classes: \{35, 39\} \rightarrow 30-40
  - Leave off digits of the ZIP code: \{39106, 39114\} \rightarrow 39***

- **Suppression of tuples**: Removing of tuples that violate the \(k\)-anonymity and thus are identifiable
Control Questions

What is a database view? How are views defined?
Control Questions

- What is a database view? How are views defined?
- Are views changeable? Under which conditions?
Control Questions

- What is a database view? How are views defined?
- Are views changeable? Under which conditions?
- How can data protection be achieved in databases?
Summary

- Views to structure databases
- Problems by changes via views
- Right system in SQL-DBS
- Privacy-aspects