Views

- Predefined queries for computation of derived tables can be declared in an SQL schema as well:

- Views are defined in a separate `CREATE VIEW` statement, simply assigning a name to a query (formulated in SQL-DML), e.g.:

  ```sql
  CREATE VIEW metropolis AS
  ( SELECT ID, Name, Inhabitants, Country 
    FROM city 
    WHERE Inhabitants >= 1000 ) ;
  ```

- According to the latest edition of the SQL standard, views may even refer to themselves. Such views are called recursive. In this case, the keyword RECURSIVE has to be given in front of VIEW.

- Recursive views are very useful for traversing data representing graphs such as maps or hierarchies (e.g., „Find all connections from X to Y of arbitrary length!“)
Queries over views

- Queries involving a view are interpreted by expanding the view name, i.e. by textually replacing it by the query associated with it in the view definition:

  e.g.:

  ```sql
  CREATE VIEW C4-profs AS
  ( SELECT Name, Dept
    FROM professors
    WHERE Rank = 'C4' )

  SELECT Name
  FROM C4-profs
  WHERE Dept = 'III'

  SELECT X.Name
  FROM ( SELECT Name, Dept
         FROM professors
         WHERE Rank = 'C4' ) AS X
  WHERE X.Dept = 'III'
  ```

- Note that this technique does no longer work for recursive views, as expansion would never terminate!
Why Recursion?

A famous and most noble family will serve as our example application, . . .

http://www.britroyals.com/royalfamily.htm
Who is the ancestor of whom?

Elizabeth \(\infty\) Philip

Diana \(\dagger\) \(\infty\) Charles
  \(\infty\) Camilla

Anne \(\infty\) Mark
  \(\infty\) Timothy

Sarah \(\infty\) Andrew

Sophie \(\infty\) Edward

William

Henry

Peter

Zara

Beatrice

Eugenie

Louise

James
Recursive Queries (1)

Who is the ancestor of whom?

- For traversing a tree (or graph) with an unknown number of edges, recursive expressions are needed.
- In order to be able to define a recursive expression, the respective queries must be explicitly named and stored: CREATE VIEW.
- Since SQL:1999 recursive queries are supported by the standard in view definitions:

```sql
CREATE RECURSIVE VIEW Ancestor AS
```
recursive sub-calls

WITH RECURSIVE Ancestor(Anc, Desc) AS
(SELECT Father AS Anc,
    Name AS Desc
FROM Parents)
UNION
(SELECT Mother AS Anc,
    Name AS Desc
FROM Parents)
UNION
(SELECT Parents.Father AS Anc,
    Ancestor.Desc AS Desc
FROM Parents, Ancestor
WHERE Parents.Name = Ancestor.Anc)
UNION
(SELECT Parents.Mother AS Anc,
    Ancestor.Desc AS Desc
FROM Parents, Ancestor
WHERE Parents.Name = Ancestor.Anc)
Recursive Queries (3)

Who is the ancestor of whom?

The recursive expression can be simplified by introducing the concept of being a parent first:

```
CREATE VIEW Parent AS
 (SELECT Father AS Anc, Name FROM Parents)
UNION
 (SELECT Mother AS Anc, Name FROM Parents)
```

```
WITH RECURSIVE Ancestor(Anc,Desc) AS
 (SELECT  Par    AS Anc,
         Name   AS Desc
         FROM   Parent)
UNION
 (SELECT Parent.Par     AS Anc,
         Ancestor.Desc  AS Desc
         FROM Parent, Ancestor
         WHERE Parent.Name = Ancestor.Anc)
```

(Georg IV, Elizabeth)
(Elizabeth, Charles)
(Philip, Charles)
(Charles, William)
(Charles, Henry)
...
In contrast to non-recursive queries, recursive ones have to be iteratively evaluated:

```
WITH RECURSIVE Ancestor(Anc,Desc) AS
    (SELECT ...)

0-te Iteration
(Georg IV,Elizabeth)
(Elizabeth,Charles)
(Charles, William)
(Charles, Henry)
...

1-te Iteration
(Georg IV,Elizabeth)
(Elizabeth,Charles)
(Charles, William)
(Charles, Henry)
(Georg IV,Charles)
(Elizabeth, William)
(Elizabeth,Henry)
...

2-te Iteration
(Georg IV,Elizabeth)
(Elizabeth,Charles)
(Charles, William)
(Charles, Henry)
(Georg IV,Charles)
(Elizabeth, William)
(Elizabeth,Henry)
(Georg IV,William)
(Georg IV, Henry)
...
Recursion is needed in order to determine the transitive closure of a given relation:

**Arbitrary graph (cyclic)**

**Edge relation**

<table>
<thead>
<tr>
<th>von</th>
<th>nach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Path relation**

transitive closure = reachability relation = “all possible paths”
Recursive Queries are difficult ...

- Obviously, the evaluation of recursive queries is expensive.

- Properties of the evaluation process:
  - Termination?
  - Number of iterations needed?
  - How to avoid the re-derivation of tuples from previous round?
  - How to estimate the size of the resulting table?
  - ... 

- The semantics of recursive expressions including negation is not always intuitively defined:
  
  Who is not ancestor and descendant for himself?
Recursive Queries and Negation

WITH RECURSIVE Ancestor(Anc, Desc) AS
    (SELECT Par AS Anc, Name AS Desc
    FROM Parent)
UNION
    (SELECT Parent.Par AS Anc, Ancestor.Desc AS Desc
    FROM Parent, Ancestor
    WHERE Parent.Name = Ancestor.Anc AND
    NOT EXIST (SELECT *
    FROM Ancestor AS A1
    WHERE A1.Anc = Ancestor.Desc AND
    A2.Desc = Ancestor.Anc))

Problem: In order to evaluate the NOT EXISTS-block, the ancestor relation should be already completely determined. This relation, however, is part of the definition of ancestor?

Who is not ancestor and descendant for himself?
Common SQL Errors

```
SELECT Name
FROM Royals
WHERE Died <> NULL
```

- **IS NOT**
  - No valid Boolean conditions for rows from Royals.

```
SELECT Name
FROM Royals
WHERE COUNT(Titel)=2
```

- **Ambivalent: R1.Name or R2.Name**

```
SELECT Name
FROM Royals AS R1, (SELECT * FROM Royals) AS R2
WHERE R1.Born=R2.Born
```

- **Subquery may not return in a unique result**

```
SELECT Name
FROM Royals
WHERE Born>(SELECT R.BORN FROM Royals AS R WHERE R.Born<1964)
```

```
SELECT Name, "1023" AS Nr, (SELECT COUNT(*) FROM Royals)
FROM Royals
WHERE Born=1996
```

- **No name specified for additional result column (e.g. AS ct)**
Summary

**SQL Foundations**
- SELECT clause
- Inner and outer joins
- Set operations
- Case distinctions

**Modularization of Queries**
- Sub queries
- Nested Table Expressions
- Scalar Full Select

**Data Types**
- Build-in data types
- Type casting

**Data Definition and Data Manipulation**
- INSERT, UPDATE, DELETE, MERGE
- Tables
- Integrity Constraints
- Views
Foundations of Information Systems

3 Database Design
Database Design
**Database Design: Principles**

**Goal**
- Modeling a part of the “real world” through abstraction
- Abstract model allows to store and process “real world” data and to answer questions about the “real world” using a computer

**Procedure**
- Requirement analysis and design of an abstract model (conceptual and external layer)
- Transformation of the abstract model (e.g. UML or Entity Relationship Model) in a specific database model (additionally: internal layer)
Abstract design standard for a DBS, first proposed in 1975

Three-level architecture

- Separate the users’ view(s) of the database from the way that it is physically represented
- Changes to one view should not affect others
- Hides the physical storage details from users
- Users should work with the data itself, without concern for how it is physically stored
- Database administrator (DBA) should be able to change the database storage structures without affecting the users’ views
- The internal structure of the database should be unaffected by changes to the physical aspects of the storage (e.g. changeover to a new disk)
- DBA should be able to change the conceptual structure of the database without affecting the users
There are three different types of database schema corresponding to the three levels in the ANSI-SPARC architecture

- **External schema**
  - A user's view of the database describes a part of the database that is relevant to a particular user
  - It excludes irrelevant data as well as data which the user is not authorized to access
  - There may be many external schemas for a given database

- **Conceptual schema**
  - Describes all the data items and relationships between them
  - Together with integrity constraints, there is only one conceptual schema per database

- **Internal schema**
  - Contains definitions of the stored records, the methods of representation, the data fields, and indexes
  - There is only one internal schema per database

*Data independency:* A change of data definition in one level does not affect the definitions in the level’s above
Life cycle of a database

1. Data processing requirements
2. Requirement analysis
3. Conceptional database design
4. Logical database design
5. Physical database design
6. HW/OS requirements

- Information requirements
- Model properties
- DBMS properties

- Information requirements
- Model properties
- DBMS properties
(1) Requirements analysis

- Analysis of the
  - Data
  - Data relations
  - Transactions
- Non-functional requirements
  - Performance requirements (response times etc.)
  - Security
  - HW/SW platforms

(2) Conceptual design

- ER or UML modeling
  - Formal description of the data objects and their relationships
  - Specification of integrity constraints
(3) Logical Design

- Goal: Creation of the conceptional and the external schema
- Transformation of a semantic model into DBS specific data model
  - e.g. ER model $\rightarrow$ relational schema
  - Determine primary keys
  - Definition of integrity constraints and transaction in a DBS specific data model (e.g. relational queries)
- Specification of view definitions (external views)
- Specification of active components (e.g. trigger)
- Assigning access authorization
(4) Physical Design

- Goal: Set up the database by creating the internal schema
- Concrete implementation of the conceptional schema
- Concrete value of domain for each attribute, e.g. CHAR(30), VARCHAR(20), NUMBER(4), NUMBER(7,2), DATE, ...
- Creating relations and loading data
- Creating views, users, access rights, integrity constraints, index structures

(5) Maintenance, Modification, Extension

- Working with the database, removing bugs, etc.
- Implementing and extending applications
- Re-Engineering, integration of additional data sources (schema and data transformation)
Entity Relationship Model
> Life cycle of a database

- **Requirement analysis**
- **Conceptional database design**
- **Logical database design**
- **Physical database design**

**Data processing requirements**

**Information requirements**

**HW/OS requirements**

**Model properties**

**DBMS properties**

**Information requirements**

**Model properties**

**DBMS properties**
The Entity Relationship Model allows us to sketch database schema designs

- Proposed by Peter Chen (1976)
- Abstract and conceptual representation of data
- Includes some constraints, but not operations
- Later: convert E/R designs to relational DB designs

Components

- **Entity** = Real-world object distinguishable from other objects, described by a set of attributes
- **Entity set** = collection of similar entities
  - All entities in an entity set have the same set of attributes
  - Each entity set has a key
  - Each attribute has a domain
- **Attribute** = property of (the entities of) an entity set
  - Attributes are simple values, e.g. integers or character string
  - Key attributes which uniquely define an entity of an entity sets, e.g. ISBN of a book
- **Relationship** = Real-world relation between entities
- **Relationship set** = how (the entities of) an entity set are related

A data model, called the entity-relationship model, is proposed. This model incorporates some of the important semantic information about the real world. A special diagrammatic technique is introduced as a tool for database design. An example of database design and description using the model and the diagrammatic technique is given. Some implications for data integrity, information retrieval, and data manipulation are discussed.

The entity-relationship model can be used as a basis for unification of different views of data: the network model, the relational model, and the entity set model. Semantic ambiguities in these models are analyzed. Possible ways to derive their views of data from the entity-relationship model are presented.

Keywords and Phrases: database design, logical view of data, semantic of data, data models, entity-relationship model, relational model, Data Base Task Group, network model, entity set model, data definition and manipulation, data integrity and consistency

1. INTRODUCTION

The logical view of data has been an important issue in recent years. Three major data models have been proposed: the network model [2, 3, 7], the relational model [8], and the entity set model [25]. These models have their own strengths and weaknesses. The network model provides a more natural view of data by separating entities and relationships (to a certain extent), but its capability to achieve data independence has been challenged [8]. The relational model is based on relational theory and can achieve a high degree of data independence, but it may lose some important semantic information about the real world [12, 15, 21]. The entity set model, which is based on set theory, also achieves a high degree of data independence, but its viewing of values such as “0” or “red” may not be natural to some people [25].

This paper presents the entity-relationship model, which has most of the advantages of the above three models. The entity-relationship model adopts the more natural view that the real world consists of entities and relationships. It
**E/R Diagram**

*Entity-relationship diagram*
- Graphical representation of the E/R model

- **Entity set**

- **Attribute** connected to its entity set; key attributes underlined

- **Relationship** connects two or more entity sets

*Example*
- Entity set **Beers** has two attributes, **name** and **manf** (manufacturer)
- Each **Beers** entity has values for these two attributes, e.g. (Bud, Anheuser-Busch)
E/R Diagram – Another Example

- **Bars**
  - name
  - addr
  - license

- **Beers**
  - name
  - manf

- **Sells**
  - Bars sell some beers
  - Drinkers frequent some bars

- **Likes**
  - Drinkers like some beers

- **Frequents**
  - Drinkers frequent some bars

**Entities**:
- **Bars**
- **Beers**
- **Drinkers**

**Relationships**:
- **Sells**
- **Frequents**
- **Likes**
The current “value” of an entity set is the set of entities that belong to it

- Example: the set of all bars in our database

The “value” of a relationship is a relationship set, a set of tuples with one component for each related entity set

- For the relationship Sells, we might have a relationship set like:

<table>
<thead>
<tr>
<th>Bar</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe’s Bar</td>
<td>Bud</td>
</tr>
<tr>
<td>Joe’s Bar</td>
<td>Miller</td>
</tr>
<tr>
<td>Sue’s Bar</td>
<td>Bud</td>
</tr>
<tr>
<td>Sue’s Bar</td>
<td>Pete’s Ale</td>
</tr>
<tr>
<td>Sue’s Bar</td>
<td>Bud Lite</td>
</tr>
</tbody>
</table>
Multiway Relationships

Relationship that connects more than two entity sets.

- Suppose that drinkers will only drink certain beers at certain bars.
- Our three binary relationships Likes, Sells, and Frequents do not allow us to make this distinction.
- But a 3-way relationship Preferences would
Relationship Types

one-to-one (1:1) relation
- Each entity of either entity set is related to at most one entity of the other set
- Example: a beer cannot be made by more than one manufacturer, and no manufacturer can have more than one best-seller

one-to-many (1:m) relation
- Each entity of the second set is connected to at most one entity of the first set, but an entity of the first set can be connected to zero, one, or many entities of the second set
- Example: A drinker has at most one favorite beer, but a beer can be the favorite of any number of drinkers, including zero

many-to-many (n:m) relation
- An entity of either set can be connected to many entities of the other set
- Example: A bar sells many beers, a beer is sold by many bars