Empty tables (and not so empty ones)

- How does an empty table look like in SQL?

- In set theory, „empty“ means: without elements. Thus, an empty table does not contain any row.

- Don‘t confuse this with a table containing just one row the fields of which all consist of NULL values – such a table is not (really) empty!

- In the datasheet view of Access the difference is clearly visible:

<table>
<thead>
<tr>
<th>City</th>
<th>River</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-empty table, consisting of a „NULL-row“</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City</th>
<th>River</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty table not containing any row</td>
<td></td>
</tr>
</tbody>
</table>
> Boolean queries in SQL

- How to "simulate" a yes/no-query in SQL?
  
  e.g.: Is there a city with more than 4 million inhabitants?

- With table queries, only an indirect answer is possible:
  
  An empty answer table is interpreted as „no“.

```
SELECT Name
FROM city
WHERE Inhabitants > 4000 ;
```

- More reasonable, but not (yet) possible as a „stand-alone“ query according to the SQL standard:

```
CHECK EXISTS (SELECT Name FROM city WHERE Inhabitants > 4000 )
```
There is a rather awkward formulation of this query (ab)using the concept of a table query as follows:

\[
\begin{align*}
\text{(SELECT 'Yes' AS Answer} \\
\text{FROM city} \\
\text{WHERE EXISTS (SELECT * FROM city} \\
\text{WHERE Inhabitants > 4000))} \\
\text{UNION} \\
\text{(SELECT 'No' AS Answer} \\
\text{FROM city} \\
\text{WHERE NOT EXISTS (SELECT * FROM city} \\
\text{WHERE Inhabitants > 4000))}
\end{align*}
\]

Even though this still is a table query, it at least appears like a yes/no query:

<table>
<thead>
<tr>
<th>Answer</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
**Case Distinction**

**Case**

- Applied in `SELECT`- or `GROUP BY` clauses

- Example:

```sql
SELECT P_NAME,
    CASE WHEN P_SIZE < 10 THEN 'SMALL'
         WHEN P_SIZE < 20 THEN 'NORMAL'
         ELSE 'BIG'
    END AS SIZE
FROM PART
```

<table>
<thead>
<tr>
<th>P_NAME</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>goldenrod</td>
<td>SMALL</td>
</tr>
<tr>
<td>frosted orange turquoise dim chocolate</td>
<td>NORMAL</td>
</tr>
<tr>
<td>royal lace plum spring coral</td>
<td>BIG</td>
</tr>
</tbody>
</table>

- Notice: Order of conditions is important!
Query Modularization
Query Modularization

Query modularization
- Sub queries (WHERE)
- Nested Table Expression (FROM)
- Scalar Full Select (SELECT, WHERE, HAVING)

Sub queries
- Returns a relation as result
- Used in the WHERE clause to determine the existence of tuples
- Syntax
  WHERE EXISTS
  | <expr> [NOT] IN
  | <expr> < | > | ... [ALL | SOME | ANY]
  (<subquery>)
**Uncorrelated sub queries**

- \[
\text{SELECT N\_NAME FROM NATION WHERE N\_REGIONKEY NOT IN (SELECT R\_REGIONKEY FROM REGION WHERE R\_NAME <> 'EUROPE')}
\]

<table>
<thead>
<tr>
<th>N_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRANCE</td>
</tr>
<tr>
<td>GERMANY</td>
</tr>
<tr>
<td>ROMANIA</td>
</tr>
<tr>
<td>RUSSIA</td>
</tr>
<tr>
<td>UNITED KINGDOM</td>
</tr>
</tbody>
</table>

- \[
\text{SELECT LASTNAME, JOB, SALARY FROM EMPLOYEE WHERE SALARY > SOME (SELECT SALARY FROM EMPLOYEE WHERE JOB='MANAGER')}
\]

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>JOB</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAAS</td>
<td>PRES</td>
<td>152750</td>
</tr>
<tr>
<td>THOMPSON</td>
<td>MANAGER</td>
<td>94250</td>
</tr>
<tr>
<td>KWAN</td>
<td>MANAGER</td>
<td>98250</td>
</tr>
<tr>
<td>GHEYER</td>
<td>MANAGER</td>
<td>80175</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Query Modularization (3)

- `SELECT LASTNAME, JOB, SALARY FROM EMPLOYEE
  WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE
  WHERE JOB='MANAGER')`

<table>
<thead>
<tr>
<th>LASTNAME</th>
<th>JOB</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAAS</td>
<td>PRES</td>
<td>152750</td>
</tr>
</tbody>
</table>

Correlated sub queries

- `SELECT R_NAME FROM REGION WHERE NOT EXISTS (SELECT * FROM NATION
  WHERE N_REGIONKEY = R_REGIONKEY AND N_NAME LIKE 'A%')`

<table>
<thead>
<tr>
<th>R_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIA</td>
</tr>
<tr>
<td>EUROPE</td>
</tr>
<tr>
<td>MIDDLE EAST</td>
</tr>
</tbody>
</table>
Nested Table Expression

- Returns a relation as result
- Used in **FROM** clause as input relation
- Must be named (**AS**)  

Example

```
SELECT N_NAME FROM NATION AS T1,
    (SELECT R_REGIONKEY FROM REGION
     WHERE R_NAME = 'EUROPE') AS T2
WHERE T1.N_REGIONKEY = T2.R_REGIONKEY
```

<table>
<thead>
<tr>
<th>N_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRANCE</td>
</tr>
<tr>
<td>GERMANY</td>
</tr>
<tr>
<td>ROMANIA</td>
</tr>
<tr>
<td>RUSSIA</td>
</tr>
<tr>
<td>UNITED KINGDOM</td>
</tr>
</tbody>
</table>
> Query Modularization (5)

**Scalar Full Select**
- Returns a single value
- Used in projection and selection part of a query

**Example**
- `SELECT C_NATIONKEY, COUNT(*) AS PER_NATION, (SELECT COUNT(*) FROM CUSTOMER) AS TOTAL,` 
  `FROM CUSTOMER` 
  `GROUP BY C_NATIONKEY`

<table>
<thead>
<tr>
<th>C_NATIONKEY</th>
<th>PER_NATION</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>1500</td>
</tr>
<tr>
<td>1</td>
<td>107</td>
<td>1500</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1500</td>
</tr>
<tr>
<td>3</td>
<td>46</td>
<td>1500</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Types
Data Types

**Standard Data Types**
- e.g. INTEGER, FLOAT, DOUBLE PRECISION, DECIMAL(precision, scale), CHAR(n), VARCHAR(n), BIT(n), DATE, TIME, ...
- Atomic/no structure
- Can be evaluated by the database system

**Large Objects (LOB, large object)**
- e.g. CLOB for text data, BLOB for binary data
- Semantics hidden from database system, e.g., movies, images, ...
- Cannot be analyzed by database system, e.g. no comparison operators (<, =, >)
- Often stored separately

**Structured Data Types**
- XML types, Object-relation types
Type casting

- Type conversion can be applied explicitly
- **CAST**-Operator
  - Syntax: `CAST(<value> AS <data-type>)`
- Also conversion between numerical data types and strings possible

- Example
  - `SELECT 1/2, CAST(1/2 AS DOUBLE), CAST(1 AS DOUBLE)/2, 1.0/2`

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

*implicite type conversion (INTEGER → DOUBLE)*
Distinct Types

**User-Defined Distinct Type (UDDT)**

- Data types defined by users
- But: not compatible with source data type (*strong typing*)
- Additionally restricts the work with data types → safer (e.g. BLOB)

```
CREATE DISTINCT TYPE <name> AS <source-data-type>
```

- Source data type must be a build-in data type
- **WITH COMPARISONS** enables comparisons
  - Must be quoted for standard data types
  - Forbidden for LOB/DATALINK

- Automatic creation of type casting functions
  - `<source-type-name> (<type-name>)`
  - `<type-name> (<source-type-name>)`

- Deletion of UDDT’s: **DROP DISTINCT TYPE** `<name>`
  - Only successful if the data type is not in use (e.g. in a table or view)
Distinct Types (2)

Example

- CREATE DISTINCT TYPE IMAGE_ID_T AS INTEGER
  WITH COMPARISONS;
- CREATE DISTINCT TYPE IMAGE_T AS BLOB;

CREATE TABLE IMAGES (  
  ID IMAGE_ID_T,
  IMAGE IMAGE_T
);

- SELECT ID+1 FROM IMAGES \(\rightarrow\) Error
- SELECT INTEGER(ID)+1 FROM IMAGES \(\rightarrow\) OK
- SELECT IMAGE FROM IMAGES WHERE ID=1 \(\rightarrow\) Error
- SELECT IMAGE FROM IMAGES WHERE ID=IMAGE_ID_T(1) \(\rightarrow\) OK