Dirk Habich

Scalable Data Management (SDM)

2

Data Models – Hands-on Relational DBs and SQL

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

DBMS Sandwich

Applications

(Relational) Database Management System (DBMS)

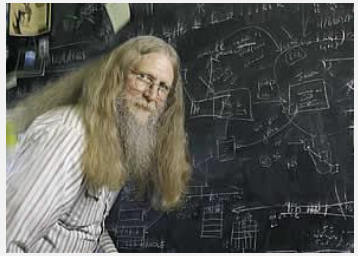
Hardware

Why Relational Databases???

Dresden Database Research Group

"Relational Databases are the foundation of western civilization."

Bruce Lindsay,
IBM Fellow @ IBM Almaden Research Center



Relational Database Management Systems (DBMS) are

- Established state-of-the-art and well researched
- Universally applicable due to its flexibility
- Available in all sizes and at all prices
- Supported by many tools

Common Concepts

- Data Modeling, Recovery, Transaction processing, Query optimization

TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

Relational Model

The Relational Model

Dresden Database Research Group

The Relational Model

- Developed by Codd (IBM) in 1970
- Considered ingenious but impractical in 1970
- Conceptually simple
- Computers lacked power to implement the relational model
- Today, microcomputers can run sophisticated relational database software

Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is

[Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.]

Relational Model – Foundations

Dresden Database Research Group

Foundations

- Domains: Integer, String[20], Date, …
  - A domain D is a set of atomic values with a specific data type
- Relation **R** is defined on a relational schema RS
  - Relational Schema RS: Set of attributes $\mathbf{\{A_1,\ldots,A_k\}}$
  - Attribute $\mathbf{A_j}$: Domain $\mathbf{D_j = dom(A_j)}$
  - Relation: Subset of the cartesian product of the domains

$$\mathbf{R \subseteq D_1 \times D_2 \times \ldots \times D_k,\ k \geq 1}$$

Example

- Domain $D_1 = \{a,b,c\}, D_2 = \{0,1\}$
- Cartesian product $D_1 \times D_2 = \{ (a,0), (a,1), (b,0), (b,1), (c,0), (c,1) \}$
- Possible relations $R_1 = \{ (a,0), (b,0), (c,0), (c,1) \}$ or $R_2 = \varnothing$

Relational Model – Foundations (2)

Dresden Database Research Group

Foundations (cont'd)

- Tuple: Element of a relation
- Cardinality of a relation: number of tuples in a relation
- Degree of a relation $R \subseteq D_1 \times D_2 \times \ldots \times D_k$: $k \geq 1$

- Representation of a relation as table
  - Attribute names A and B are column names
  - Order of rows (tuples) and columns (attributes) within a table is important

R_1	A	B
	a	0
	b	0
	c	0
	c	1

Relational Model – Foundations (3)

Dresden Database Research Group

Example

Cities	Name	Inhabitants	State
	Munich	1.353.186	Bavaria
	Dresden	523.058	Saxony

Relational schema

- {Name, Inhabitants, State} with
  dom(Name) = String[40],
  dom(Inhabitants) = INTEGER and
  dom(State) = String[40]}

Instances

- t_1[Name] = Munich, t_1[Inhabitants] = 1.353.186 and t_1[State] = Bavaria
- t_2[Name] = Dresden, t_2[Inhabitants] = 523.058 and t_2[Germany] = Saxony

In general

- Database schema = Set of relational schemas
- Database = Set of relations

Primary Key Concept

Dresden Database Research Group

Notation

- Given a relational schema RS and $X \subseteq RS$, then t[X] denotes the tuple restricted to the attributes in X

Primary Key

- A primary key is a special column (or combination of columns) designated to uniquely identify each tuple within a relational schema (RS)
- A set of attributes $X \subseteq RS$ is called a primary key, if following conditions are met:
  - **Uniqueness:** for all relations R of the relational schema RS applies that $\forall\, t_i, t_j \in R: \; t_i[X] = t_j[X] \Rightarrow i = j$
  - **Definedness:** $\forall\, t_i \in R: \; t_i[X] \neq$ 'NULL'
  - **Minimality:** $\neg\exists\, Z \subset X\ (Z \neq X)$, still satisfying the conditions above

R	PNr	Salary
	1	1700
	2	2172
	4	1700

Please note

- Key property also applies to data that will be stored in the relation in the future
- Key candidates: Several subsets of a relational schema may qualify as key (one has to be chosen)
- Attributes of the key are underlined

Foreign Key Concept

Dresden Database Research Group

Foreign Key

- A foreign key is a column or group of columns in a relational table that provides a link between data in two relational schemas.
- $Y \subseteq RS_1$ is foreign key of a relation R_1 with reference to a relation R_2, if following conditions are met:
  - $X \subseteq RS_2$ is primary key in R_2
  - **Definedness:** $\forall\, t_i \in R_1 : (t_i[Y] = \text{'NULL'} \vee \exists\, t_j \in R_2 : t_i[Y] = t_j[X])$
  - **Minimality:** $\neg\exists\, Z \subset Y\ (Z \neq Y)$, so that the first two conditions are satisfied

R_1	PNr	Salary	Department
	1	1700	IT
	2	2172	Sales
	4	1700	*NULL*

R_2	Name	Building
	IT	A
	Sales	B
	Development	A

Database Design

Database Design: Principles

Dresden Database Research Group

Goal

- Modeling a part of the “real world” (also called “mini-world”) through abstraction
- Model allows to store and process “real world” data and to answer question about the “real world” using a SQL-system

Procedure

- Step 1: Requirement analysis and design of an abstract model (e.g., UML or Entity Relationship Model)
- Step 2: Transformation of the abstract model into a specific relational database schema
- Step 3: Create relational database schema within a SQL system
- Step 4: Load data into SQL system
- Step 5: Work with the SQL system (update data; insert data; delete data; process SQL queries)

Relational Database Schema

Dresden Database Research Group

Definition

- A set S of relational schemas that belong the the same database
- S is the name of the whole database schema
- $S = \{RS_1, RS_2, RS_3, ..., RS_n\}$
- $RS_1, RS_2, RS_3, ..., RS_n$ are the names of the individual relational schemas within the database S

Example

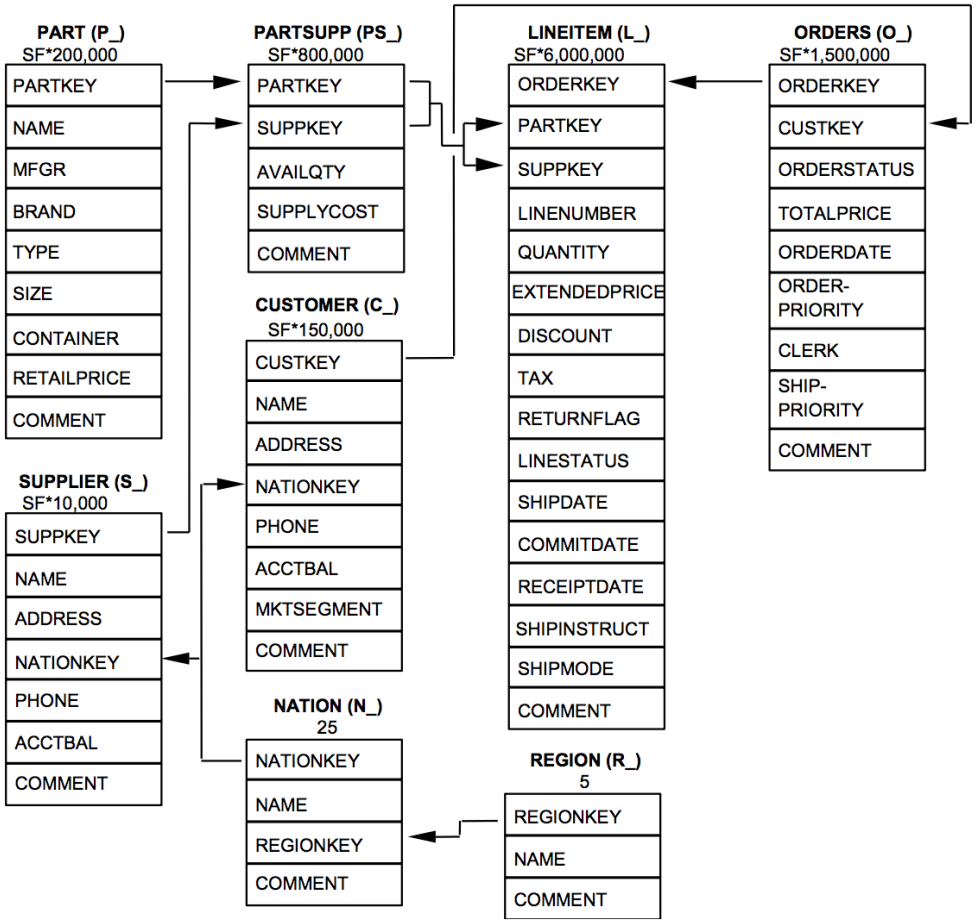
TPC-H Relational Database Schema

- Is a Decision Support Benchmark
- TPC-C database consists of 8 relational schemas (tables)
  - Each with its own attributes
  - Each with its own primary key
  - Tables are connected via foreign keys (edges in the figure)

Conclusion:

- All data has to be modelled as relational data

Figure 2: The TPC-H Schema



Relational Algebra (Operational Perspective)

Motivation

Dresden Database Research Group

The meaning of relational algebra

- Formal language for the calculation method for query results
- Internal representation for DB queries (basis for optimization)
- Not visible to the user of a DBMS

Limits

- Relational algebra does not contain operations
  - to create or delete relations
  - to insert, delete and modify tuples
- Exclusively “reading“ operations
- It is assumed that the relations of the database already somehow exist and have been filled with tuples.

Relationale Algebra

Dresden Database Research Group

Algebra

- Given a set N: set of relations
- Operations op_j: $N^k \rightarrow N$ (seclusion)

Basic operations of relational algebra

- Five basic operations (projection, selection, cartesian product, union, and difference)

- Given two relations $R(A_1,...,A_r)$ and $S(B_1,...,B_s)$ with degree r and degress s then the following operations

$$R' := < op >_{<Parameters>} (R) \text{ as well as}$$
$$R'' := R < op >_{<Parameters>} S$$

reveal relations again

Basic Operation - Projection

Dresden Database Research Group

Definition: projections

Let A‘ be a subset of the attributes of a relation R(A_1, …, A_n).
The projection of the attributes A‘ from a tupe t ∈ R is defined as the tuple

$$\pi_{A'}(t) = (A'_1(t), \ldots, A'_m(t))$$

The projection of attributes A' of a relation R is defined as the relation

$$\pi_{A'}(R) = \{\pi_{A'}(t) | t \epsilon R\}$$

Illustrative

- Projection is an operation that selects certain columns from a relation and output them as a new relation
  → duplicates in the output relations are dropped by default

Basic Operation - Projection/Example

Dresden Database Research Group

Students

StudentID	LastName	FirstName	MiddleName	Birthday	City	SgNr	Amount
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400

$$\pi_{Lastname,City}(Students)$$

LastName	City
Schmidt	Würzburg
Meisel	Schweinfurt
Schmidt	Rimpar
Krause	Würzburg
Schäfer	Kitzingen
Rasch	Würzburg
Bakowski	Schweinfurt

Basic Operation - Selection

Dresden Database Research Group

Definition: Selection

- The selection of a relation R is defined as the set of all tuples from R that fulfill the selection condition P:

$$\sigma_P(R) = \{t | t \epsilon R \wedge P(t)\}$$

- P consists of
  - Operands: Constants or name of an attribute
  - Comparison operation: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$
  - Boolean operators: $\wedge$, $\vee$, $\neg$

Illustrative

- Selection is an operation that selects certain rows from a relation and outputs them as a new relation

Basic Operation – Selection/Example

Dresden Database Research Group

Students

StudentID	LastName	FirstName	MiddleName	Birthday	City	SgNr	Amount
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400

$$\sigma_{LastName='Schmidt'}(Students)$$

StudentID	LastName	FirstName	MiddleName	Birthday	City	SgNr	Amount
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0

Basic Operation – Cartesian Product

Dresden Database Research Group

Cartesian Product: $R \times S$

- The cartesian product of two relations $R(A_1, A_2, \ldots, A_n)$ und $S(B_1, B_2, \ldots, B_m)$ is defined as a relation

$$R \times S = \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) \mid (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$$

A_1	A_2
1	A
2	B
3	C

X

B_1	B_2
1	X
2	Y
4	Z

=

A_1	A_2	B_1	B_2
1	A	1	X
1	A	2	Y
1	A	4	Z
2	B	1	X
2	B	2	Y
2	B	4	Z
3	C	1	X
3	C	2	Y
3	C	4	Z

Weitere Basisoperationen

Dresden Database Research Group

Union: $R \cup S$

- $R \cup S := \{ r \mid r \in R \text{ or } r \in S \}$

- R and S have the same schema
- Domains of attributes must be compatible

Difference: $R - S$

- $R - S := \{ r \mid r \in R \text{ und } r \notin S \}$

- R and S have the same schema
- Domains of attributes must be compatible

Advanced Operation: Theta- und Equi-Join

Dresden Database Research Group

Theta-Join (Verbund): $R \bowtie_{\Theta} S$

- Selecting specific tuples from the Cartesian product R×S:

$$R \bowtie_{i\Theta j} S := \sigma_{A_i \Theta B_j}(R \times S)$$

- with $\Theta \in \{=, \neq, <, \leq, >, \geq\}$

Equi-Join: Theta-Join mit Θ gleich "="

- Beispiel: $R \bowtie_{A=E} S$

R	A	B	C	D
	1	2	3	4
	4	5	6	7
	7	8	9	0

S	E	F	G
	1	2	3
	7	8	9

$R \bowtie_{A=E} S$	A	B	C	D	E	F	G

SQL – Foundations (Frontend for users)

SQL

SQL (Structured Query Language)

- Descriptive Query Language

Consists of

- DQL (*data query language)*
- DML (*data manipulation language*)
- DDL (*data definition language*):
- DCL (*data control language*): access right management

SQL Standard

- SQL/86, SQL/89, SQL/92 (=SQL2), SQL/99, SQL:2003 (=SQL3), SQL:2006
- Part 1: SQL/Framework (Standard Description)
- ***Part 2: SQL/Foundation***
- Part 3: SQL/CLI (*call level interface*)
- Part 4: SQL/PSM (*persistent storage modules*)

SELECT Statement

Dresden Database Research Group

SELECT Statement

```
SELECT [DISTINCT] <attribute-list>      → Projection, Duplicate elim.
FROM <table-name> [AS <alias>], …       → Cartesian Product
WHERE <predicate-list>                  → Selection on tuple level
GROUP BY <attribute-list>               → Grouping
HAVING <predicate-list>                 → Selection on group level
ORDER BY <attribute> [ASC|DESC],…       → Sorting
```

Select Conditions Examples

- Pattern search in Strings
- Range search
- NULL-Werte

```
WHERE R_NAME [NOT] LIKE 'A%'
WHERE R_REGIONKEY [NOT] BETWEEN 1 AND 3
WHERE R_COMMENT IS [NOT] NULL
```

SELECT Statement (2)

Dresden Database Research Group

Example (SAMPLE)

- Name and salary of employees belonging to department A00

```
SELECT FIRSTNME, LASTNAME, WORKDEPT, SALARY
FROM EMPLOYEE
WHERE WORKDEPT = 'A00'
```

- Result

```
FIRSTNME     LASTNAME     WORKDEPT     SALARY
-----------  -----------  -----------  ---------
CHRISTINE    HAAS         A00          52750
VINCENZO     LUCCHESSI    A00          46500
SEAN         O'CONNELL    A00          29250
```

SELECT Statement (3)

Dresden Database Research Group

Example (SAMPLE)

- Average salary per department

```
SELECT DEPTNAME, AVG(SALARY) AS AVG_SALARY
 FROM DEPARTMENT D, EMPLOYEE E
 WHERE E.WORKDEPT = D.DEPTNO
 GROUP BY DEPTNAME
```

aggregation function
(e.g. COUNT, SUM, MIN, MAX)

- Result

```
DEPTNAME                    AVG_SALARY
--------------------------  ----------
ADMINISTRATION SYSTEMS      25153,33
INFORMATION CENTER          30156,66
MANUFACTURING SYSTEMS       24677,77
OPERATIONS                  20998
PLANNING                    41250
SOFTWARE SUPPORT            23827,5
SPIFFY COMPUTER SERVICE DIV.  42833,33
SUPPORT SERVICES            40175
```

SELECT Statement (4)

Dresden Database Research Group

Example (TPCH)

- Total revenue per country, if this value is over 50 Mio.; sorted

```
SELECT N_NAME, SUM(O_TOTALPRICE) AS TURNOVER
FROM ORDERS, CUSTOMER, NATION
WHERE O_CUSTKEY = C_CUSTKEY
  AND C_NATIONKEY = N_NATIONKEY
GROUP BY N_NAME
HAVING SUM(O_TOTALPRICE)>50000000
ORDER BY N_NAME
```

- Result
  -

```
N_NAME                    TURNOVER
------------------------  ------------
ARGENTINA                 129997977,11
EGYPT                     66482178,24
JORDAN                    273941626,19
MOROCCO                   1093739712,6
```

Join Operations

Dresden Database Research Group

Inner join (Theta- or Equi-Joins)

- Default join operation
- Only tuples with join partners are part of the result
- Example
  -
```
SELECT E.Name, D.Name, D.Boss
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.Department = D.Name
```
  -
```
E.Name    D.Name          D.Boss
-------   ------------    --------
Paul      Administration  Paul
Dirk      Production      Dirk
```

Employee	
Name	Department
Paul	Administration
Fritz	NULL
Dirk	Production

Department	
Name	Boss
Administration	Paul
Production	Dirk
IT	NULL

Join Operations (2)

Dresden Database Research Group

Outer join

- Tuples without join partner are part of the result
- Attributes of the other relation are filled with **NULL** values

Left outer join

- Tuples of the left relation appear in the join result
- Example

```
SELECT E.Name, D.Name, D.Boss
FROM EMPLOYEE E LEFT OUTER JOIN DEPARTMENT D
     ON E.DEPARTMENT = D.Name

E.Name   D.Name        D.Boss
-------  ------------  --------
Paul     Support       Paul
Dirk     Production    Dirk
Fritz    -             -
```

Employee	
Name	Department
Paul	Administration
Fritz	NULL
Dirk	Production

Department	
Name	Boss
Administration	Paul
Production	Dirk
IT	NULL

Join Operations (3)

Dresden Database Research Group

Right outer join

- Tuples of the right relation appear in the join result
- Example

```
SELECT E.Name, D.Name, D.Boss
FROM EMPLOYEE E RIGHT OUTER JOIN DEPARTMENT D
     ON E.DEPARTMENT = D.Name

E.Name   D.Name        D.Boss
-------  ------------  --------
Paul     Support       Paul
Dirk     Production    Dirk
-        IT            -
```

Employee	
Name	Department
Paul	Administration
Fritz	NULL
Dirk	Production

Department	
Name	Boss
Administration	Paul
Production	Dirk
IT	NULL

Join Operations (4)

Dresden Database Research Group

Full outer join

- All tuples appear in the join result

```
SELECT E.Name, D.Name, D.Boss
FROM EMPLOYEE E FULL OUTER JOIN DEPARTMENT D
     ON E.Department = D.Name

E.Name   D.Name        D.Boss
-------  ------------  --------
Paul     Support       Paul
Dirk     Production    Dirk
Fritz    -             -
-        IT            -
```

Employee	
Name	Department
Paul	Administration
Fritz	NULL
Dirk	Production

Department	
Name	Boss
Administration	Paul
Production	Dirk
IT	NULL

Set Operations

Dresden Database Research Group

Set operation

- Applied to relations with the same schema
- **UNION**, **EXCEPT**, **INTERSECT**
- Set (e.g. **UNION**) vs. multiset (**UNION ALL**) semantic
- Example

```
SELECT R_NAME FROM REGION
UNION
SELECT 'ANTARCTICA' FROM SYSIBM.SYSDUMMY1

1
-----------
AFRICA
AMERICA
ANTARCTICA
ASIA
EUROPE
MIDDLE EAST
```

Case Distinction

Dresden Database Research Group

CASE

- Applied in **SELECT**- or **GROUP BY** clauses
- Example

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

```
P_NAME                                    SIZE
----------------------------------------  -------
goldenrod                                 SMALL
frosted orange turquoise dim chocolate    NORMAL
royal lace plum spring coral              BIG
…
```

- Notice: Order of conditions is important!

Data Types

Data Types

Standard Data Types

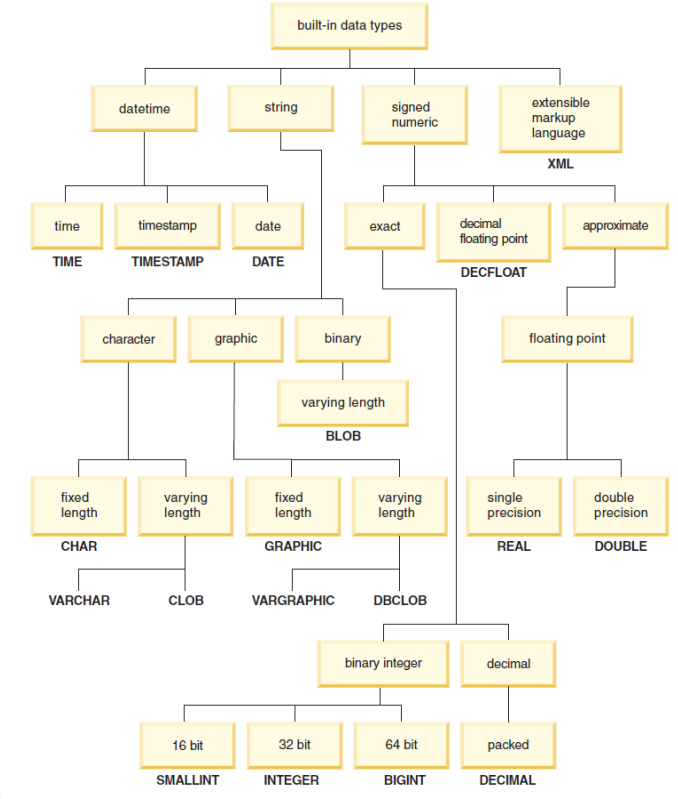
- e.g. INTEGER, FLOAT, DOUBLE PRECISION, DECIMAL(precision, scale), CHAR(n), VARCHAR(n), BIT(n), DATE, TIME, …
- Atomic/no structure
- Can be evaluated by the database system

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

Dresden Database Research Group

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

1      2      3      4
----   ----   ----   ----
0      0      0.5    0.5
```

INTEGER

DOUBLE

implicite type conversion
(INTEGER → DOUBLE)

Data Manipulation (DML) and Data Definition (DDL)

SQL-DML

Inserting Tuples

```
INSERT INTO REGION VALUES (6,'Antarctica','')
```

```
INSERT INTO NATION (N_NATIONKEY, N_NAME, N_REGIONKEY)
SELECT NATIONKEY, NAME, 6 FROM ANTARCTIC_NATIONS
```

Updating Tuples

```
UPDATE LINEITEM
SET L_DISCOUNT = L_DISCOUNT + 0.01
WHERE L_SUPPKEY = 12
```

Deleting Tuples

```
DELETE FROM REGION WHERE R_REGIONKEY > 5
```

SQL-DML (2)

Dresden Database Research Group

Merging Data (MERGE)

- Combination of **INSERT**, **DELETE** and **UPDATE** (SQL:2003)
- Merges two relations in a predetermined manner
- **MERGE INTO** <table1>
  USING <table2>|<fullselect> **ON** <condition>
  WHEN [**NOT**] **MATCHED THEN** <operation>
- Parameters
  - <table1>: table which should be updated
  - <table2>: new data
  - <condition>: tuple comparison (when are two tuples equal?)
  - **MATCHED**: Tuples exist in both relations
  - **NOT MATCHED**: Tuple appear only in the new relation
  - <operation>: Action (**UPDATE**, **INSERT**, **DELETE**)

- (Not fully supported in H2)

SQL-DML (3)

Dresden Database Research Group

Example (MERGE)

- Two relations: PERSONS (left) and NEW_PERSONS (right)

```
NAME      SALARY              NAME      SALARY
-------   ---------           -------   ---------
Paul      2000
Fritz     1000                Fritz     1500
                              Dirk      3000
```

- Merging both relations

```
MERGE INTO PERSONS P
USING NEW_PERSONS N ON P.NAME = N.NAME
WHEN MATCHED THEN
     UPDATE SET P.SALARY = N.SALARY
WHEN NOT MATCHED THEN
     INSERT (NAME,SALARY)
     VALUES (N.NAME, N.SALARY)
```

Result (PERSONS)

```
NAME      SALARY
-------   ---------
Paul      2000
Fritz     1500
Dirk      3000
```

Database Objects

Dresden Database Research Group

Relation (`TABLE`)

- Unordered set of tuples
- Permanent or temporary

View (`VIEW`)

- Virtual relation
- e.g. simplifiction of queries, user-defined view on data

Index (`INDEX`) → Discussed next week

- Primary or secundary
- Data structure to store / find data
- Additional: preserving uniqueness and sorting

SQL-DDL

Dresden Database Research Group

Definition of Database Objects (DDL)

- e.g. table
- Creating using **CREATE** command

```
CREATE TABLE REGION (
        R_REGIONKEY INTEGER NOT NULL PRIMARY KEY,
        R_NAME CHAR(25) NOT NULL,
        R_COMMENT VARCHAR(152)
)
```

- Deleting using **DROP**

```
DROP TABLE REGION
```

- Changing using **ALTER**

```
ALTER TABLE REGION ADD COLUMN AREA INT
```

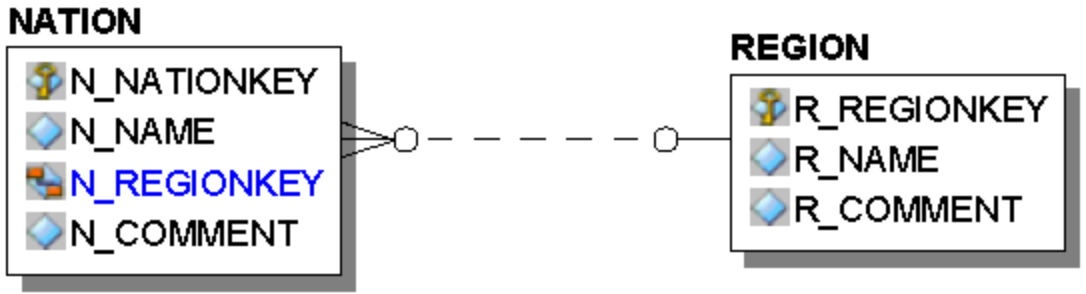
SQL-DDL (2)

Dresden Database Research Group

Integrity Constraints (*constraints*)

- Primary Key, Foreign Key, Distinct values (**UNIQUE**), value-based (**CHECK**)
- **ALTER TABLE** <table> **ADD CONSTRAINT** <constraint-name>
  - **PRIMARY KEY** (<attribute-list>)
  - **FOREIGN KEY** (<attr-list>) **REFERENCES** <table>(<attr-list>)
  - **UNIQUE** (<attribute-list>)
  - **CHECK** (<predicate>)
- Example

```
ALTER TABLE REGION
ADD CONSTRAINT MAX5
CHECK (R_REGIONKEY
BETWEEN 1 AND 5)
```



SQL-DDL (3)

Dresden Database Research Group

View (VIEW)

- Virtual relation to simplify queries and to define user-specific representations
- Also used for security reasons to hide tuples or attributes
- Specified by SQL queries
- Example

```
CREATE VIEW NAT_REG AS
SELECT N_NAME, R_NAME
FROM NATION, REGION
WHERE N_REGIONKEY = R_REGIONKEY

SELECT * FROM NAT_REG
```

N_NAME	R_NAME
ALGERIA	AFRICA
MOZAMBIQUE	AFRICA
MOROCCO	AFRICA
…	

Database Transactions

Transactions Overview

Dresden Database Research Group

Transactions

- A **transaction** is the execution of a **sequence** of one or more **operations** (e.g., SQL queries) on a shared database to perform some higher-level or complex function
- It is the basic unit of change in a relational DBMS (SQL system): Partial transactions are not allowed!

Example

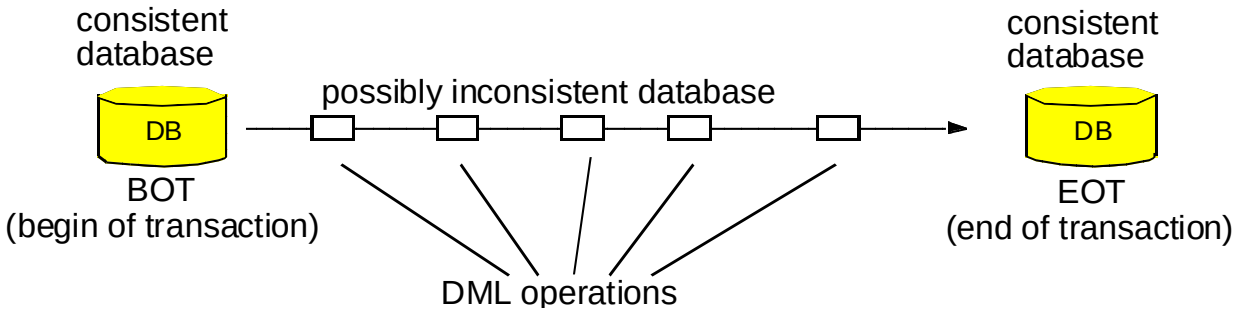
- Move $100 from Andy's bank account to his bookie's account
- Transaction
  - Check whether Andy has $100
  - Deduct $100 from his account
  - Add $100 to his bookie's account

Transaction Management

Dresden Database Research Group

Principle of a transaction

- Sequence of successive DB operations that transform a database from a consistent state into another consistent state
  surrounded by: BOT …….. EOT (Commit / Abort)



- Properties
  - ACID: Atomicity, Consistency, Isolation, Durability
  - A transaction will always come to an end
  - Normal (commit): changes are permanently stored within the DB
  - Abnormal (abort / rollback): already composed changes are taken back
- Note: EOT state must not be different from BOT state

ACID Properties of Transactions

Dresden Database Research Group

Atomicity

- Indivisibility due to the transaction definition (Begin - End)
- All-or-nothing principle, i.e., the DBS guarantees
  - Either the complete execution of a transaction …
  - … or the ineffectiveness of the whole transaction (and of all associated operations)

Consistency

- A successful transaction guarantees that all consistency requirements (integrity requirements) have been met

Isolation

- Multiple transactions run isolated from each other and do not use (inconsistent) intermediate results from other transactions

Durability

- All results of successful transactions have to be made persistent

Summary

Relational Model

- Foundations
- Primary Key/Foreign Key
- Relational Algebra

SQL Foundations

- SELECT clause
- Inner and outer joins
- Set operations
- Case distinctions

Data Types

- Build-in data types
- Type casting

Data Definition and Data Manipulation

- INSERT, UPDATE, DELETE, MERGE
- Tables
- Integrity Constraints
- Views

Transactions

- Definition of a transaction
- ACID Properties

Homework

The Relational Model

Dresden Database Research Group

The Relational Model

- Developed by Codd (IBM) in 1970
- Considered ingenious but impractical in 1970
- Conceptually simple
- Computers lacked power to implement the relational model
- Today, microcomputers can run sophisticated relational database software

Read the Original Paper

- 02-codd.pdf in OPAL (folder papers)

Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. CODD
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. DATA DEPENDENCIES IN PRESENT SYSTEMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is

[Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.]