Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Stephan Heidinger

Datenbanken und Informationssysteme
Sommersemester 2011
28. Januar 2012

Find any errors? Please send them back, I want to keep them!

Abstraction

- Three kinds of data abstraction: External schema → Conceptual schema → Internal Schema → Stored Data
- **Conceptual Schema:** simplified view of DB, modeling abstractions, semantic constraints.
- **Internal/Physical Schema:** *(Implementation)* Indexing, Division of tables among disks, storage management, Placement of new rows in tables

Abstraction: applications dont care about how it works exactly, as they can access the db in the same way.

Abstraction: later modification of db (addition of information) should not lead to modification of apps

- **External Schema/View:** subset of whole db, security reasons, need-to-know

State vs Schema

- **Database Schema:** formal definition of the structure of dbs contents, defined, when db is created
- **Database State:** *(Instance of the Schema)* Contains actual data, changes frequently
- data is structured in tables (relations) with name, namend columns (attributes) and set of rows (tuples)
- db queries refer to conceptual schema, dbms translates into execution plan that refers to internal schema, optimizes (more complex queries ⇒ bigger savings)
- **declarative query languages:** describe what information is sought, not how (⇒ SQL)

Transaction Management

- *virtual machine illusion*, isolation of concurrency, fault tolerance, atomic commit, consistency checking, distributed coordination

ACID: *(atomicity, consistency, isolation, durability)* sequences of db commands are executed as **atomice unit**, support for backup an recovery, support for concurrent users

Database Management System (DBMS)

- generic, application independent software system implementing a data model
- definition of DB schema, storage of an instance of this schema, querying the instance, changing the state, defining users and privilieges
- **Data Model:** formal languages: DDL *(data definition language, declaring database schema)*, (S)QL *(querying the current database state)*, DML *(database manipulation language, changing the state)*

i.e.: Relational Model, Entity Relationship Model, Object-Oriented Models, UML, XML

- application programs process users request, formulate sql statements, present the result in a user-friendly way
- **DB applications Systems:** users access database concurrently, dbms runs as background server processes,
- **Database users:**

  Database Administrator (DBA): know about all schemas, change conceptual and internal schema, gives acces rights to users, security, monitors performance/disk space/etc, backups, recoveries

  Application Programmer: writes programs for use by naïve users (safe data entry, report generation, data browsing), knows sql/programming languages well, supervised by DBA, might do conceptual design

  sophisticated User: knows something of sql, may use SQL-console/generic db-tools, non-standart aggregations/evaluations w/o help from programmers

  naïve user: uses db only via applictaion programs, often unaware of db itself, data entry, browsing, external views

- **Database tools:**(interactive) sql console, graphical/menu-based query-tools, interface for db-access for standart programming languages, report generators, web interface, import/-export/backup/recovery/performance monitoring/... tools

Information Modelling

...the process of revising a (formal) model of (parts of) the real world.

- ...Herauslassen von details, Konzentrieren auf die „wichtigen“ Sachen
- meist „Bilder anstatt Sprache“

Relational Model & SQL Introduction

- database → set of tables; schema defines: names of tables, columns of each table, integrity constraints
- Table ≡ relation, Row ≡ tuple, Column ≡ attribute, Data type of column ≡ domain of attribute, Table entry ≡ attribute value
- **key:** always identifies a single row in a table (keys are contraints)
- **foreign key:** no physical pointer in relational model, foreign keys „point“ to other tables, foreign key and reference may have different names

- **bad relational tables:** redundancy, many empty cells, several real world concepts in one table
- **SQL:** SELECT `<attr>` FROM `<table>` WHERE `<condition>`
  SQL ist case-insensitive, außer in strings, wildcards (use LIKE): % - sequence, _ single character, remove duplicates: DISTINCT, sorting: ORDER BY [ASC|DESC]

Entity-Relational Model

- **DB design hard:** Expertise (designer has to know the field he is designing or needs an expert, which may be a WDG), Flexibility (exceptions, corner cases, ...), Size
  $\Rightarrow$ DB design as a *multi-step* process.
- **Three Phases of DB Design:** (partitioning the problem, attack one problem after another, DBMS does not influence conceptual design, can change DBMS later on

  Conceptual Database Design: Produces initial model (conceptual data model, ER model)

  Logical Database Design: transform conceptual model into DBMS-conform model (relational model)

  Physical Database Design: indexes, table distribution, buffer size, etc $\rightarrow$ maximize performance
- **ER Diagrams:** concept for easy design, good for communication with non-experts

  Entity: an object in the real world, must be distinguishable from each other (identity)

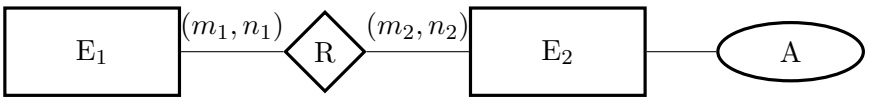
  Relationship: between pairs of entities

  Attribute: property or feature of an entity/relationship, the **value** is a data type (printable representation)

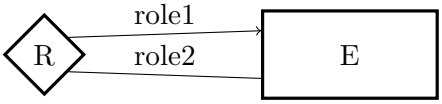
  entity type: set of similar entities

  relationship type: set of similar relationships

  Cardinality: E_1 is related to min m_1, max n_1 items of E_2, * means no limit, weaker cardinalities: (a, b) weaker than $(c, d) \Leftrightarrow a \leq c \wedge b \leq d$



- **relationship between entities of the same type**: edges need role names (i.e. precondition and knowledge of)



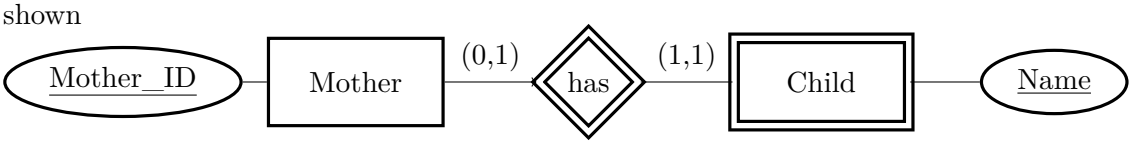
- **cardinalities:**

  many-to-many: $\square - (0, *) - \diamondsuit - (0, *) - \square$ or $\square - N - \diamondsuit - M - \square$: requires an extra table

  one-to-many: $\square - (\_, *) - \diamondsuit - (\_, 1) - \square$ or $\square - 1 - \diamondsuit - N - \square$: do not require an extra table

  one-to-one: $\square - (\_, 1) - \diamondsuit - (\_, 1) - \square$ or $\square - 1 - \diamondsuit - 1 - \square$
- **key:** uniquely identifies the entity/relationship, composite keys allowed, keys not required in er, but required in relational schema, add artificial identifiers (ie numbers)
- **weak entitiey:** *detail* entities that cannot exist without a *master*/owner, cardinality (1,1)

on detail side, key of master is inherited, keys always composite, inherited part of key not shown



- **n-ary relationship:** relationships may connect more than 2 entities
- **association entity:** turns n-ary relationships into weak entities with multiple owners
- **supertype/Generalisation:** (i.e. mechanic is an employee) → child inherits attributes of parent, not shown in ER though
- **translation into realtional model:**
  - create **table** for each entity
  - **columns** are attributes
  - **primary key** of the table is primary key of entity, if none available, add artificial key
  - **weak entities**: 1table with key of owner are foreign keys
  - **one-to-many:** key of „one“ side added as foreign key into „many“ table, minimum value shows, if null values are allowed or not
  - **many-to-many:** create new table, keys of both entities, foreign composite primary key
  - **one-to-one:** proceed like one-to-many, add key, where it makes more sense (i.e. null values can be enforced), no null values needed on both sides: one may have the attribute or extra table, if no null values on both sides $(1,1) - (1,1)$: merge tables
  - **supertype/Generalisation:**
    1. each entity type is mapped into seperate relation, superkey is foreign primary, each table holds only specific attributes, access requires joins, minimum redundancy
    2. one relation for all types, lots of NULL values, additional distinguishing attributes (i.e. jobkind) may be needed, no joins
    3. independent relations for subtypes, generalisation lost, redundancy in schemas, redundancy in tuples, supertype relation needed

beyond ER and relational modelling

- **object-orientation:**
  attributes → *visibile internal structure* of entity (so ER does not provide encapsulation) or *observer method* (so ER does provide encapsulation
  mutator → cannot be expressed in ER models (in SQL-3 possible)
  Aggregation (hirarchy) → „part of“ relationships
  Association (hirarchy) → „member of“ relationship
- **UML:** (Unified Modeling Language) mostly graphical, ïndustry standard“ for language independent: analysis of requirements, visualisation of problems, design of programs/databases, communication with application/domain experts, implemenentation, documentation
  class: defines properties (attributes) and behaviour (methods), contains classname, attribute definition, method definition
  cardinalities: a,b → exactly a or b, a..b → between and including a and b, a..* → at least a (or combinations)

Relational Normal Forms

functional dependencies (FD) generalisation of keys, things that depend on the key, form: $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$

cover α^+: „a set of attributes α is the set of all attributes B that are uniquely determined by the attributes α (with respect to a given FD set $\mathcal{F}$):

$$\alpha_{\mathcal{F}}^{+} := \{B | \mathcal{F} \text{ implies } \alpha \rightarrow B\}$$

Algorithm:
$x \leftarrow \alpha$;
`while` x did change `do`
`foreach` given FD $\alpha_i \rightarrow \beta_i$ `do`
`if` $\alpha_i \subseteq x$ `then`
$x \leftarrow x \cup \beta_i$;
`fi`
`od`
`od`
deutsch: Die transitive Hülle beeinhaltet alle transitiven Paare (entweder direkt, oder durch transitive Verkettung)

all possible keys: $\alpha \subseteq \mathcal{A}$ is key of $R \Leftrightarrow \alpha_{\mathcal{F}}^{+} = \mathcal{A}$
oder: ein key ists dann, wenn die komplette Menge die trans. Hülle ist.

minimal keys: start with empty set: $x = \emptyset$
x may never contain a rhs B of FDs $\alpha \rightarrow B$ when x already contains the lhs ($\alpha \subseteq x$) rechts darf nichts rein, was links schon drin ist
As long as x does not determine alls attributes of R, choose any remaining attribute $X \in \mathcal{A} - x^+$
Do both: add X to x and for each FD $\alpha \rightarrow X$ add α to x
backtrack for non-minimal keys and for alternative minimal keys

Insert Anomalies: unrelated concepts are stored in same table, ie instructor cannot be inserted without a course

deletion anomalies: unrelated concepts are stored in same table, ie when the last course of an instructor is delted, the instructor is deleted

update anomalies: transitive data in the same table
ie when one emp switches proj, all other emp in this proj do too

Employee (unnormalized)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

First Normal Form (1NF) all table entries (attribute values) are atomic
no repeating columns (attributes) within a row, no multi-valued columns
queries and sorting become easier

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

Second Normal Form (2NF) Functional Dependence: all attributes functionally depend on primary key:
non-depending stuff → new table
prevents insert/deletion anomalies

Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

Skills (2NF)

emp_no	skills
1	C
1	Perl
1	Java
2	Linux
2	Mac
3	DB2
3	Oracle
3	Java

Third Normal Form (3NF) standard relational form for use
every FD $A_1, \ldots . A_n \rightarrow B$ satisfies at least one of:

- the FD is trivial
- the FD follows from a key, because $\{A_1, \ldots, A_n\}$ or subset of it is already a key of R
- B is a key attribute (element of a key of R)

or: remove transistive dependence something is not dependent in the key into subtables
prevents update anomalies

Employee (3NF)

emp_no	name	dept_no
1	Kevin Jacobs	201
2	Barbara Jones	224
3	Jake Rivera	201

Department (3NF)

dept_no	dept_name
201	R&D
224	IT

Determinante: Attributmenge, von der andere Attribute funktional abhängen

Schlüsselkandidat: Menge von Attributen, von der alle Attribute der Relation voll funktional abhängig sind.

Boyce Codd Normal Form (BCNF) more restrictive than 3NF, easier to define, short: „all FDs are already enforced (represented) by keys“
anders: BCNF ist erfüllt, wenn (3NF gilt,) jede Determinante Schlüsselkandidat (oder die Abhängigkeit trivial) ist.

Fourth Normal Form (4NF) nur triviale Mehrwertige Abhängigkeiten enthalten, oder die MwA gehen von Superschlüsseln aus.
kurz: Es darf nicht etwa n-mal Attribut a und unabhängig davon m-mal Attribut b vom Schlüsselwert abhängen.
in 4NF überführen: jede MwA in extra Tabelle auslagern

decomposition: If R is not in BCNF, split it into two tables along the violating FD:
If $A \rightarrow B$ violates BCNF remove B from R and add new relation (table) $A \rightarrow B$

lossless decomposition: if the intersection of the attributes of the new tables is a key of at least on of them
the original relation can be reconstructed via a natural join

Normalisation

minimal cover: want a minimal cover, $\mathcal{F}'$ with same transitive closure $(\mathcal{F}')^+ = \mathcal{F}^+$

- Canonicalize rhs: Replace FD $\alpha \to B_1, \ldots, B_m$ by corresponding set of FDs: $\alpha \to B_i, 1 \leq i \leq m$
- Minimize lhs: For each FD $A_1, \ldots, A_n \to B$ and each $i = 1, \ldots, n$ check, if $B \in \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n\}^+_{\mathcal{F}}$, if so $\mathcal{F} \leftarrow (\mathcal{F} - \{A_1, \ldots, A_n \to B\}) \cup \{A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \to B\}$
- Remove implied FDs: For each FD $\alpha \to B$, check if $B \in \alpha^+_{\mathcal{F}-\{\alpha \to B\}}$, if so: $\mathcal{F} \leftarrow \mathcal{F} - \{\alpha \to B\}$

3NF synthesis algorithm produces lossless decomposition into 3NF and preserves FDs

- Compute minimal cover of given FDs $\mathcal{F}'$
- For each lhs in $\mathcal{F}'$ create relation with attributes: $\mathcal{A} = \alpha \cup \{B | \alpha \to B \in \mathcal{F}'\}$
- if none of these relations contain a a key of original relation R add one relation containing the attributes of a key of R
- For any two relations $R_{1,2}$ constructed, if the schema of R_1 is contained in R_2 drop R_1

a „good“ ER will result in a high normal form