

Course Notes on The Bases of the Relational Model

Bases of the Relational Model: Summary

- Relations
- Constraints
- Update operations
- Data manipulation (later chapters)
 - ◇ algebra
 - ◇ tuple relational calculus
 - ◇ domain relational calculus
 - ◇ SQL

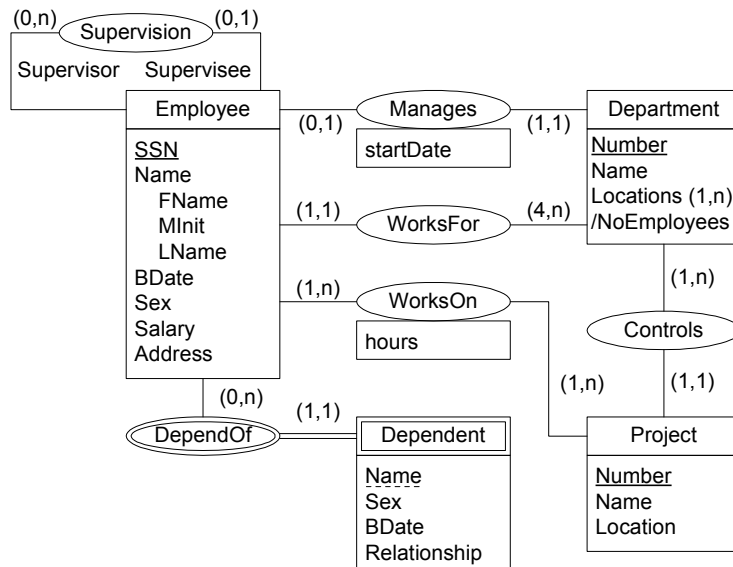
Intuitive View of Relations

- **Popular view** of the relational model = information is structured as 2-dimensional **tables** of simple values (with lines, or rows, or tuples, and columns, or attributes)
- **Relations** are tables with some restrictions
 - ◇ the order of rows is immaterial
 - ◇ the order of columns is immaterial
 - ◇ relations have no duplicate rows \Rightarrow each relation has one or more key
- A relation can also be seen as a **predicate**, i.e., a set of properties, assertions
- **Database** = collection of relations, but
 - ◇ relational data structures are richer than tables
 - ◇ a data model is not just data structures but also operations for manipulating the data structures

2

- The order of rows and columns is immaterial = manipulating (querying or updating) relations cannot depend or rely on relations being ordered
- In fact, a relation is a set of tuples (row or lines in a table)
- Sets are unordered \Rightarrow the tuples in a relation are unordered (there is no notion of “first” tuple or “next” tuple)
- If the output of a program or of a query must be ordered, this should be explicitly requested in the program or query
- The elements of a set are all distinct \Rightarrow there are no duplicate tuples in a relation (but SQL allows duplicate tuples)
- A set may be implemented as a file (i.e., as some kind of table or sequential structure), but this is a physical implementation, and the order of records in a file (or of rows in a table) is not be visible to users, i.e., may not be exploited in application programs (physical data independence)

ER Schema for the Company Example



3

Relational Schema for the Company Example

Employee								
<u>SSN</u>	FName	LName	BDate	Address	Sex	Salary	SuperSSN	DNo

Department			
<u>DNumber</u>	DName	DMgr	MgrStartDate

DeptLocations	
<u>DNumber</u>	<u>DLocation</u>

Project			
<u>PNumber</u>	PName	PLocation	DNumber


WorksOn		
<u>PNo</u>	<u>ESSN</u>	Hours

Dependent				
<u>ESSN</u>	DependentName	Sex	BDate	Relationship

4

- In terms of entities and relationships:
 - ◇ there are 4 entities represented as relations Employee, Department, Project, and Dependent
 - ◇ DeptLocations represents a multivalued attribute of Department
 - ◇ WorksOn represents a M-N relationship between Employee and Project
 - ◇ Dependent merges a relationship and a less important entity

Relational Data Model

- A **data model** provides mechanisms (languages) for defining
 - ◇ data structures
 - ◇ operations for retrieval and modification
 - ◇ integrity constraints
- The **relational data model** provides 
 - ◇ mechanisms for defining domains and the structure of relations
 - ◇ several data-manipulation languages (DML): relational algebra, domain relational calculus, tuple relational calculus, update operations
 - ◇ mechanisms for specifying particular constraints (e.g., key, referential integrity) + a language for specifying arbitrary constraints

Relation Schema and Relation Value

- **Relation** = Schema + Value
- **Relation schema:** $R(A_1 : D_1, \dots, A_n : D_n)$
 - ◊ R = name of the relation
 - ◊ **domains** (D_1, \dots, D_n) = sets of atomic values; domains in R are not necessarily all distinct
 - ◊ **attributes** A_1, \dots, A_n , all distinct in R
 - ◊ **structure** $(A_1 : D_1, \dots, A_n : D_n)$ (i.e., the data type of relation tuples)
- **Tuple:** $\{A_1 : d_1, \dots, A_n : d_n\}$ = a set of $\langle \text{attribute}, \text{value} \rangle$ pairs
- **Relation value:** set of tuples $\subseteq \{ \{A_1 : d_1, \dots, A_n : d_n\} \mid d_i \in D_i \}$

6

- Some notations leave domains implicit: $R(A_1, \dots, A_n)$ (but domains must be specified in the relation schema, i.e., CREATE TABLE in SQL)
- Some notations leave attributes implicit: $\{\langle d_1, \dots, d_n \rangle \mid d_i \in D_i\}$, but attributes are essential because the same domain can appear several times in the same relation
 When attributes are omitted in the notation, the idea is that relation columns are ordered (i.e., that relation columns can be identified by their position): this is not the correct definition of the relational model
- A precise definition of the basis of the relational model:
 - ◊ relations, and algebraic and calculus operations on them can be naturally formalized in terms of an indexed Cartesian product of domains = $A_1 : D_1 \times \dots \times A_n : D_n$
 - ◊ the value of a relation = a subset of the indexed Cartesian product
 - ◊ see, e.g., *A Precise Definition of Basic Relational Notions and the Relational Algebra*, A. Pirotte, ACM SIGMOD Record, 13-1, 1982, pp. 30-45

Domains

- Domains carry important structural information
- Domains define comparability of values: attribute values can only be meaningfully compared if they range on the same domain (= typing checking in programming languages)
- Domains have been underused: in SQL and RDBMSs in general, domains are restricted to the data types of traditional programming languages (e.g., integer, real, character) and date
- In more modern languages (e.g., object-oriented), domains are application dependent (e.g., employee names, salaries)

7

- Relations are value-based: elementary values are the smallest units of information
- Domain = a named set of atomic values, a set of possible values for an attribute
- Domains could be viewed as conceptual user-defined data types (e.g., part numbers, city names, person names, dates, weights), carrying more application semantics than their representation at a lower level as traditional data types (e.g., integer, real, character)
- For example
 - ◇ city names and person names may both be represented as character strings
 - ◇ city names and person names can be represented as domains in the relational model (not in SQL)
 - ◇ a language that would support application-oriented domains could forbid (by type checking), for example, to compare for equality a city name with a person name (SQL cannot forbid that)
- Such comparability information is not representable in RDBMSs: it is implicit and may lead to incorrect application programs
- A richer version of domains could include
 - ◇ conversion functions, a set of applicable operations on domain values (e.g., computation, ordering) like abstract data types (ADTs)
 - ◇ operations to combine values from different domains
 - ◇ application-dependent structures not possible with RDBMSs (e.g., geometric figures, design objects): this is one advantage of object-oriented systems

Relation Keys

- **Superkey** = one (or more) attributes that (together) possess the property of **unique tuple identification**
 - ◇ their values always uniquely identify at most one tuple in the relation
 - ◇ unicity constraint = no two tuples with the same value for those attributes
- The set of all attributes of a relation is a superkey (because the value of a relation is a set of tuples)
- **Key** = minimal superkey, i.e. a group of attributes that loses the property of unique identification if any one attribute is removed from the group
- In general, a relation has several keys (**candidate keys**)
- The definition of keys is **intensional information** (i.e., it belongs to the schema)
⇒ it must be satisfied by all all legal extensions of the relation

Primary Key

- Commercial RDBMSs and SQL require that each relation have one primary key
- If a relation has several keys, one of them is privileged as primary key
- The values of a primary key should always be known (while the value of nonprimary keys may be null)
- In practice, the primary key is the most useful candidate key, which is naturally suggested by database design from the application domain (e.g., how are employees, projects identified in practice in the enterprise, by name, by number?)
- For relations that express a relation (e.g., *WorksOn*), the primary key is the combination of primary keys of entity relations involved in the relationship

9

- The concept of relation key (primary or not) should not be confused with that of indexes (also sometimes called "keys" in traditional data management)
 - ◇ a key is a semantic concept, that serves to identify objects in the application domain
 - ◇ an index is a physical concept used for performance optimization
 - ◇ a key may or may not be indexed, and an index may or may not be a key

Relational Database

- **Relational database** \approx collection of relations
- **Relational database schema** = relation schemas + integrity constraints
- **Database value, extension** = collection of relation values
- Several schemas
 - ◇ **conceptual**: information content from the real world (typically, entity-relationship or class diagrams of object models)
 - ◇ **logical**: information expressed in the data model of the DBMS (typically relational)
 - ◇ **physical**: information organized for storage media

Relational Constraints

- **Constraint** = any prescription (or assertion) on the schema (i.e., valid for all database extensions) not defined in the data-structure part of the schema
- Constraints cannot be deduced from the current extension of the database (they are part of the database schema)
- Relational constraints can be classified as
 - ◇ keys (candidate keys, primary key)
 - ◇ various dependencies (functional, multi-valued, etc.): see later
 - ◇ referential integrity
 - ◇ "ad-hoc" constraints (all the constraints specific to an application domain)

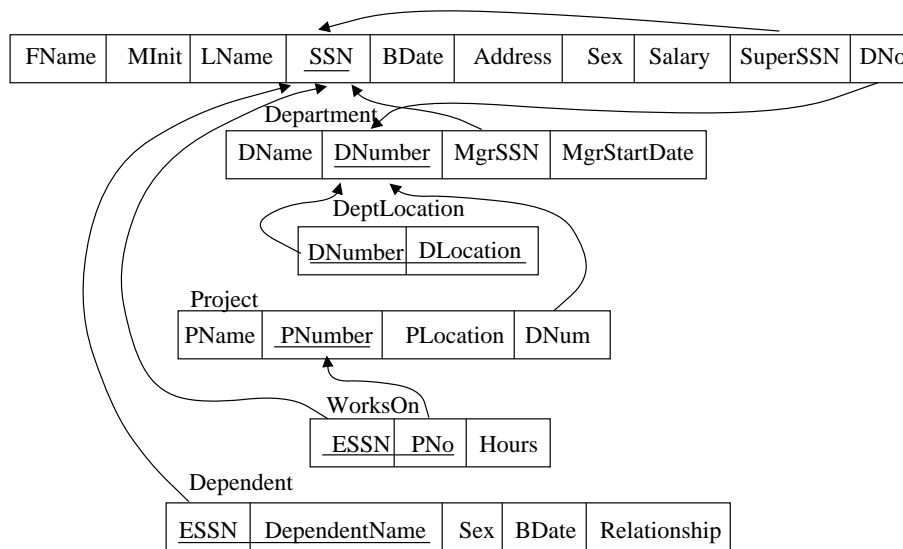
- Constraint = all that you would like to express at the level of the structure of the database (i.e., the schema) and that you cannot express with the mechanisms for structure description available in the data model
- The traditional term is “integrity constraint”, although “consistency constraint” or “schema constraint” would be better terms
- Remember that constraints are properties of data, valid for all the applications that use the data. They are part of the database schema rather than of applications (they are valid for all applications of the database)
- Some constraints have a similar form and occur frequently (e.g., referential constraints); recent versions of DBMSs allow to specify those constraints declaratively, and let the DBMS generate programs to check for possible constraint violations whenever updates to the database are attempted

Referential Integrity

- Relational constraint involving two relations R_1 and R_2 (or twice the same relation); there must exist in R_2 attribute(s) A_2 such that
 - ◇ A_2 has (have) the same domain as the primary key of R_1
 - ◇ each value of A_2 in a tuple of R_2 occurs as value of the primary key in a tuple of R_1
- Referential integrity is typically relational, ubiquitous in relational DBs
 - ◇ results from expressing links with equality of values
 - ◇ expresses comparability information (like domains) for two different attributes.
 - ◇ expresses information that is taken care of by the data structure (entities and relationships) in entity-relationship schemas

- A_2 is sometimes called foreign key in R_2
- But A_2 is not a key in R_2 , it is put in correspondence with A_1 which is a key in R_1

Referential Integrity in the Company Schema



13

- Examples:
 - ◇ $\text{Employee}[\text{DNo}] \subseteq \text{Department}[\text{DNumber}]$ (the set of values of attribute DNo in relation Department is included in the set of values of attribute DNumber of relation Department)
 - ◇ all the values of attribute ESSN of relation WorksOn must appear as values of attribute SSN of relation Employee; interpretation: all the employees that are reported to work on a project must be registered in the main list of employees in relation Employee
- Referential integrity constraints are very frequent in relational schemas
- Why are there so many referential integrity constraints in relational schemas?
 - ◇ because the relational model expresses links between pieces of information as equalities of values (example: equality between a value of SSN in a tuple of relation Employee and a value of ESSN in a tuple of WorksOn)
 - ◇ equality is a symmetric link, while what is really intended is a **pointer** from a tuple of WorksOn to a tuple of Employee.
 - ◇ thus a referential integrity constraint forces a symmetric link to be a directed link.
 - ◇ pointers are present in almost all data models (e.g., object-oriented models) but they were banned from the relational model

Adhoc Constraints

- Application-dependent constraints, of arbitrary complexity
 - ◇ **structural constraints**
 - * the salary of each employee is smaller than that of his/her supervisor
 - * department managers are employees of the department that they manage
 - * department managers are supervisors
 - * there are at least 4 employees in each department
 - ◇ **transition constraints** (concern two successive states of the database)
 - * salaries are nondecreasing
 - * the sex of an employee does not change

14

Special Names for Constraints

- Structural aspects of the relational model are presented as constraints
 - ◇ **key integrity**: each relation has a primary key + possibly candidate keys
 - ◇ **domain integrity**: attributes must obey the definition of their domain (i.e., specific set of values and of applicable operations)
 - ◇ **entity integrity**: primary-key values may not be null
 - ◇ **candidate key integrity** (= key uniqueness): key values must be unique
 - ◇ **column integrity**: a constraint that supplements domain integrity (e.g., MgrStartDate is a date after 1970)
 - ◇ **row integrity**: concerns a single tuple (e.g., if BDate is before 1950, then Salary is at least 40000)

15

Constraints versus Data Structures

- There is **no fundamental difference of nature** between constraints and data structures: the distinction depends on the power of the data-structuring mechanisms
- The same piece of information can be modeled **as fact or as constraint**, depending on its stability (e.g., headquarters are located in Houston)

16

Base and Derived Relations

- Base relations are those in the community schema, they are stored on disk
- Derived relations (views or external schemas) are defined from combining base and other derived relations by operations of the relational model
- The definition of derived relations and their correspondence with base relations are part of the database schema

17

Derived Relations: Views

- Derived relations can be presented as **views** to application programs (ANSI external schemas)
- Views are redundant and consistent with the underlying base relations
- Views simplify application programs
- Views may or may not be materialized (i.e., stored on disk): this is an efficiency issue that should be under the control of the DBMS and invisible to users
- Storing them
 - ◊ introduces physical redundancy and complicates integrity enforcement
 - ◊ accelerates querying and slows updating

18

Derived Relations: Snapshots

- Derived relations that are not synchronized at all times with base relations
- Refreshed from base relations at regular intervals
- Users of snapshots accept to work with data that is not up-to-date to gain efficiency in access times (particularly for distributed data)
- To easily establish consistency at refreshing times, snapshots may be restricted to read-only access

19

Operations of the Relational Model

- **Data Definition Language:** declare a relation, specify a constraint, define physical structures
- **Data Manipulation Language** (DML) for access and retrieval
 - ◇ algebra, domain calculus, and tuple calculus are generally considered to be part of the model
 - ◇ SQL has redefined the corresponding operations and defined some extensions, notably for aggregate functions
- **Update operations:**
 - ◇ update of tuples in the current value of a relation (insert, delete, modify)
 - ◇ update of the schema: create or delete a relation, add or suppress an attribute

20

Programming Tuple Insertion

- If insertion violates a constraint, then
 - ◇ either reject insertion
 - ◇ or correct violation
- Examples
 - ◇ tuple to be inserted has a null value for primary key: ask user for a value and proceed with insertion
 - ◇ value for a foreign key does not exist in the relation where the attribute(s) is primary key: ask user for a new tuple in that relation (possibility of cascading updates)

21

Programming Tuple Suppression

- If suppression violates a referential constraint, then
 - ◇ either reject suppression
 - ◇ or propagate suppression to the tuples that reference the suppressed tuples
 - ◇ or set to null the attribute values that reference the suppressed tuple (unless these attributes are part of the primary key)

Examples

- suppress a department \Rightarrow suppress its locations
- suppress an employee \Rightarrow suppress his/her dependents
- suppress a department \Rightarrow set to null the reference to a department in projects, until projects are assigned to another department

- Options for tuple insertion and tuple suppression can be specified declaratively in the database schema of current RDBMSs