- Are we not clever enough to come up with something better?
- The algorithms for decomposing into BCNF guarantee a lossless decomposition

  But not the preservation of FDs
- A deeper reason?
- It is provably the case that FD preservation may be impossible to achieve
- Exercise: Schema $R(J, K, L)$, with $JK \rightarrow L$ and $L \rightarrow K$

  (a) Verify that it is not in BCNF

  (b) Show that every decomposition will fail to preserve $JK \rightarrow L$

- Idea behind the decomposition algorithm for achieving BCNF:

1. If relation $R(\mathbf{A}, \mathbf{X}, Y)$ that is not in BCNF due to $\mathbf{X} \rightarrow Y$
   (that is, $Y \notin \mathbf{X}$, and $\mathbf{X}$ not a superkey)
   Then:
2. Decompose R into $R_1(\underline{\mathbf{X}}, Y)$ and $R_2(\mathbf{A}, \mathbf{X})$
   The FDs for R_2 are the "projections of the original FDs" onto the remaining attributes (see next example)
3. If one of the generated relations is not in BCNF, go back with that relation to Step 1.

Example: $R(A, B, C, D)$ with

$\mathcal{F}_0$: $A \to B, \quad B \to C, \quad CD \to A, \quad AC \to D$

It is not in BCNF due to $B \to C$ (B not a superkey)

Decompose into:

1. $R_1(B, C)$ with $B \to C$, i.e. $R_1(\underline{B}, C)$
2. $R_2(A, B, D)$ with FDs that are the projections of $\mathcal{F}_0$ onto $\{A, B, D\}$

   That is; All the FDs of the form $\mathbf{Z} \to \mathbf{W}$ that can be derived from $\mathcal{F}_0$ and $\mathbf{Z}, \mathbf{W} \subseteq \{A, B, D\}$

   $\mathcal{F}_0^+ = \{A \to B,\ B \to C,\ CD \to A,\ AC \to D,\ A \to C,\ BD \to A,\ A \to D\}$

   (plus some trivial ones)

   The projections have the "two sides" in $\{A, B, D\}$

   $A \to B,\ BD \to A,\ A \to D$

   Candidate keys for R_2: $\{A\}$ and $\{B, D\}$
3. The two schemas are in BCNF, so we stop

Exercise: (about projected FDs) In the decomposition

$Wine(Vineyard, Region, Country) \rightsquigarrow$
$Wine1(Vineyard, Region), \quad Wine2(\underline{Region}, Country)$

With FDs for $Wine$: $Vineyard, Country \to Region$
$Region \to Country$

Verify that $Vineyard \to Region$ IS NOT a projection of $Vineyard, Country \to Region$ onto schema $Wine1$

Not entailed: So, projection is not just about dropping attributes

Example: (running example, c.f. page 109) FDs as before plus:

emp_name → #emp (*)

(no repeated names)

We also have the relation:

emp_skill3(#emp, emp_name, #skill, skill_date, skill_level)

Two candidate keys: {#emp,#skill} and {emp_name,#skill}

With (*) attribute emp_name is prime: Belongs to candidate key

Schema is in 3NF: No transitive dependencies via non-prime attributes

Not in BCNF: $\{\#emp,\#skill\} \supsetneq \{\#emp\}$ and

#emp → emp_name (**)

Then, the FD (**) does not invalidate the 2NF (emp_name is prime now), but does invalidate BCNF for emp_skill3

BCNF is more demanding than 2NF

A decomposition: (based on the problematic FD)

emp_skill3.1(<u>#emp</u>, <u>#skill</u>, skill_date, skill_level)

emp_skill3.2(<u>#emp</u>, emp_name)

Final Remarks

- In the decompositions shown in this chapter, we did not care about introducing referential and foreign-key constraints

  We should do so

  Exercise: Revisit the decompositions we made and introduce those constraints where natural and expected

- Normal forms were present at the very inception of the relational model

  3NF was introduced by Codd in 1972

  BCNF by Boyce and Codd in 1974

- In practice, one usually settles for 3NF

  In most common cases, a decomposition can be found efficiently (NP-completeness of deciding 3NF refers to the worst-case)

- With 3NF, data redundancy is reduced, information is preserved under decompositions that achieve it, and dependencies too
- With BCNF there may be additional reduction of data redundancy, information is still preserved, but dependencies may not be preserved
- There are other normal forms we haven't covered
- Most prominent one among them is the 4th Normal Form (4NF)

  Introduced by Ron Fagin in 1977
- 4NF deals with a different kind of dependencies: Multi-Valued Dependencies (MVDs)

- MVDs are important to model and guarantee "independence" of attributes

- They are also used to connect DBs with the probabilistic notions of (stochastic) independence

  Quite useful in Data Science, ML, AI

  (we will come back if time permits)

- Data redundancy and updates anomalies are not independent parameters

  Data redundancy is likely to lead to update anomalies and inconsistencies

- We can concentrate on data redundancy as a "measure" of good design

  Actually, NFs are usually justified in terms of avoiding data redundancy

- For quite a long time and very surprisingly, no research provided a solid justification for these normal forms (along these lines)
- In the sense that a particular normal form is *the best one can have* considering what is achieved and missed

  There was the belief and assumption that this was the case, but no proof ....
- Any justification for these normal forms should be given in terms of "information contents" and its theory

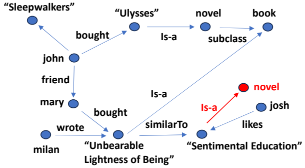
  Namely "Information Theory" (as started by C. Shannon in 1948)
- This research has been carried out quite recently by M. Arenas, L. Libkin, S. Kolahi

  - Marcelo Arenas, Leonid Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. *J. ACM*, 2005, 52(2):246-283

  - Solmaz Kolahi. Dependency-Preserving Normalization of Relational and XML Data. *J. Comput. Syst. Sci.*, 2007, 73(4):636-647

- The notion of “good design” (or well-designed schema with dependencies) is formulated in information-theoretic terms
- Now a theorem tells us that a schema is well-designed iff it is in BCNF
- If one wants to preserve dependencies, with 3NF one pays the lowest price in terms of data redundancy
- Recent applications of DBs in ML have made the “Sixth Normal Form” (6NF) popular

  Also due to use of “Graph DBs” (sets of 2-ary relations)



- Very informally, a relation schema is in 6NF when the attributes are the primary key, and at most one extra attribute

  Very commonly, 2-attribute (binary) relation schemas

- The Graph DB (or Knowledge Graph) could be strored in a relational DB
- 6NF avoids the use of null values

  If you do not have a value associated to an identifier, simply skip that entry in the table

  Helping to represent semi-structured data in structured terms
- Number of joins grows (for query-answering and other tasks)

  There are ways to handle them

  For example, what about a query asking for "the books that are similar to those bought by friends of Joe"?
- There is quite recent and relevant research on join optimization

  Still a very important research topic (and implementation efforts)

Data Management and Databases

Chapter 3: Databases and Query Languages

Leopoldo Bertossi

Universidad San Sebastián

Facultad de Ingeniería y Ciencias

General Observations

- Having covered DB design, we will go now into the logical interface of a RDBMS and RDBs
- As discussed several times, the main characteristics of RDBMSs are determined by Codd's proposal
- Codd proposed the "12 Rules of RDBs"

  Providing guidelines for building and assessing Relational DBMSs
- They are a good starting point for this chapter ...

Codd's 12 Rules for RDBs:

1. The Information Rule:

   All information in a relational database is represented explicitly at the logical level in exactly one way by values in tables.

2. Guaranteed Access Rule:

   Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a table name, primary key value, and column name.

3. Systematic Treatment of Null Values:

   Null values (distinct from empty character string or a string of blank characters and distinct from zero or any other number) are supported in the fully relational DBMS for representing missing information in a systematic way, independent of data type.

4. Dynamic On-line Catalog Based on the Relational Model:

   The database description is represented at the logical level in the same way as ordinary data, so authorized users can apply the same relational language to its interrogation as they apply to regular data.

5. Comprehensive Data Sublanguage Rule:

   A relational system may support several languages and various modes of terminal use (for example, the fill-in-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all of the following is comprehensible: data definition, view definition, data manipulation (interactive and by program), integrity constraints, and transaction boundaries (begin, commit, and rollback).

6. View Updating Rule:

   All views that are theoretically updateable are also updateable by the system.

7. High-level Insert, Update, and Delete:

   The capability of handling a base relation or a derived relation as a single operand applies nor only to the retrieval of data but also to the insertion, update, and deletion of data.

8. Physical Data Independence:

   Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representation or access methods.

9. **Logical Data Independence:**

   Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

10. **Integrity Independence:**

    Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

    A minimum of the following two integrity constraints must be supported:

    - Entity integrity: No components of a primary key is allowed to have a null value.

    - Referential integrity: For each distinct non-null foreign key value in a relational database, there must exist a matching primary key value from the same domain.

11. **Distribution Independence:**

    A relational DBMS has distribution independence. Distribution independence implies that users should not have to be aware of whether a database is distributed.

12. Nonsubversion Rule:

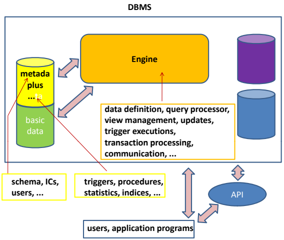
If a relational system has a low-level (single-record-at-a-time) language, that low-level language cannot be used to subvert or bypass the integrity rules or constraints expressed in the higher-level (multiple-records-at-a-time) relational language.

- In addition to the above rules, one might also add the following rule Rule Zero:

  Data Management via Relational Capability:

  For any system that is claimed to be a relational database management system, that system must be able to manage data entirely through its relational capabilities. http://itsy.co.uk/ac/0405/Sem3/44271_DDI/Lec/3_CoddsRules.htm

- The Architecture of a RDBMS:



- There may be several DBs run by the RDBMS

- A RDBMS interacts with the external computational world

  For that there are specialized "Application Program Interfaces" (APIs)

- Metadada (schema, ICs. etc.) are stored in the DB

- One can also store procedures, and triggers in particular

  There are standardized languages for specifying them (and vendors' proprietary languages)

- Integrity constraints (ICs) are prominent in Codd's Rules
  We have discussed a few classes of ICs
- Only some classes of ICs can be defined with the DB schema
  Becoming part of the schema
  And the RDBMS takes care of maintaining them (satisfied)
- Some classes of ICs cannot be declared at that stage
  And they are not automatically supported (maintained)
- This is the case of general functional dependencies
  Among FDs, only Key Constraints can be declared (and automatically maintained)
- Those non-declarable/maintainable have to be enforced and maintained from the user's side:
  - Active rules (or triggers)
  - Application programs that interact with the RDBMS
- In both cases, violation views can be useful

- Some RDBMSs (vendors) support Informational Constraints

  They are ICs declared by the user, but not checked or maintained by the DBMS

  So as every IC, they capture more semantics of the application domain

  The RDBMS assumes they are true (trusting the DB creator), and can use them

  What for?

- A RDBMS can use ICs (maintained or assumed to be true) for semantic query optimization

  Optimization of QA through the use of ICs

  In contrast with syntactic query optimization mentioned in Chapter 1

- Example: Schema: $Person(Name, Address, Job)$
  FD: $Name \rightarrow Address$ defined as informational constraint
  (this FD would not be maintainable by the RDBMS; not a key)
- Query about people with more than one address (in RC; not essential)
  $\mathcal{Q}(x):\ \exists y \exists u \exists v \exists w (Person(x, y, v) \land Person(x, u, w) \land y \neq u)$
- Answer using the FD: Empty! ($\emptyset$) No need to see the data!
- Example: $Emp(Name, Position, Project)$, $Sal(Name, Salary)$
  Range or check constraint: CEOs make more than 100K
  $\bar{\forall}(Emp(x, y, z) \land\ Sal(x, u) \land y = ceo \rightarrow u > 100K)$
- Query: Employees with salary lower than 50K
  $\mathcal{Q}'(x):\ \exists y \exists z \exists u (Emp(x, y, z) \land Sal(x, u) \land u < 50K)$
- The join for CEOs can be avoided!

- More about null values: One can declare `NOT NULL` Constraints

  Some attributes cannot take the value NULL

- A value `NULL` is used to represent a missing, unknown, non-applicable, ... datum

  The SQL Standard is unclear the semantics (meaning) of NULL values

- Different RDBMSs differ in the way they operate with NULL values, which is problematic

Emp	Name	Position	Salary	Age
	john	clerk	40 K	35
	mary	CEO	85 K	NULL
	ken	account.	60 K	40
	carol	NULL	90 K	19

  This instance satisfies the "`NOT NULL`" constraint for *Name*, but not for *Age*

- When a constraint declares a set of attributes as a key, they cannot take the value NULL

- Key constraints and "`NOT NULL`" constraints go together

- If *Name* is declared a key, it cannot take the value `NULL`
- This has to do with the way NULL values are treated by the DBMS

  It does not know if it represents a value that is equal or different from the other certain (or null) values

Emp	Name	Position	Salary	Age
	john	clerk	40 K	35
	NULL	CEO	85 K	NULL
	ken	account.	60 K	40
	carol	NULL	90 K	19

If Name is the key, it should be an identifier: How could NULL be compared with certain values, e.g. "john"?

Emp	Name	Position	Salary	Age
	john	clerk	40 K	35
	NULL	CEO	85 K	NULL
	NULL	account.	60 K	40
	carol	NULL	90 K	19

Or worse: We cannot say that two NULL values are the same, i.e. they represent the same (uncertain) value

- To all the (certain) data items, DBMSs apply the "unique names assumption" (UNA): Different names in the DB denote different outside-world objects; so they are treated as different

  The UNA does not apply to `NULL`

Relational Algebra (revisited)

- Two relations with the same schema

WINE1	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5
	130	Tokay	1980	12.5

WINE2	W#	GRAPE	VINTAGE	PERCENTAGE
	130	Tokay	1980	12.5
	140	Chenas	1981	12.7
	150	Volnay	1978	12.5

- The union of them: WINE1 $\bigcup$ WINE2

WINE3	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5
	130	Tokay	1980	12.5
	140	Chenas	1981	12.7
	150	Volnay	1978	12.5

- No duplicates, as usual in set-union

- The intersection of them: WINE1 $\bigcap$ WINE2

WINE4	W#	GRAPE	VINTAGE	PERCENTAGE
	130	Tokay	1980	12.5

WINE1	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5
	130	Tokay	1980	12.5

WINE2	W#	GRAPE	VINTAGE	PERCENTAGE
	130	Tokay	1980	12.5
	140	Chenas	1981	12.7
	150	Volnay	1978	12.5

- The difference of them: WINE1 $\setminus$ WINE2

WINE4	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5

- The difference is a “relative complement”

  That is, relative to another relation

- This keeps the result sensible and within the finite

- In RA there is only this form of limited complement

- Two relations, not necessarily with same schema

GRAPE	GRAPE	AREA	COUNTRY
	Chenas	Beaujolais	France
	Volnay	Bourgogne	France
	Chanturgues	Auvergne	France

YEAR	VINTAGE	QUALITY
	1979	Good
	1980	Average

- The product of them: GRAPE $\times$ YEAR

GY	GRAPE	AREA	COUNTRY	VINTAGE	QUALITY
	Chenas	Beaujolais	France	1979	Good
	Chenas	Beaujolais	France	1980	Average
	Volnay	Bourgogne	France	1979	Good
	Volnay	Bourgogne	France	1980	Average
	Chanturgues	Auvergne	France	1979	Good
	Chanturgues	Auvergne	France	1980	Average

- Possibly a huge table, and many combinations that do not make much sense

- The product is an expensive operation we may want to avoid
  Or apply only after we have reached smaller tables using other operations

- Usually it makes more sense from the application point of view to combine tables via a join
- Essential binary operator of RA

WINE	W#	GRAPE	VINTAGE	QUALITY
	100	Chenas	1977	Good
	200	Chenas	1980	Excellent
	300	Chablis	1977	Good
	400	Chablis	1978	Bad
	500	Volnay	1980	Average

LOCATION	GRAPE	AREA	AVG-QUALITY
	Chenas	Beaujolais	Good
	Chablis	Bourgogne	Average
	Chablis	California	Bad

- The natural join: WINE $\bowtie_{\text{GRAPE}}$ LOCATION

WL	W#	GRAPE	VINTAGE	QUALITY	AREA	AVG-QUAL
	100	Chenas	1977	Good	Beaujolais	Good
	200	Chenas	1980	Excellent	Beaujolais	Good
	300	Chablis	1977	Good	Bourgogne	Average
	300	Chablis	1977	Good	California	Bad
	400	Chablis	1978	Bad	Bourgogne	Average
	400	Chablis	1978	Bad	California	Bad

- Relations are composed via the values in common taken by attributes in common (or the same data type)
- There are other forms of join

- The join is a common but expensive operation in RDBS
- One can apply more complex join conditions
- The join above used the join condition as follows:

  $$\text{WINE} \bowtie_{\mathit{WINE.GRAPE}=\mathit{LOCATION.GRAPE}} \text{LOCATION}$$

  Values for *GRAPE* attribute in the two tables coincide
- We could also do the join:

  $$\text{WINE} \bowtie_{\mathit{WINE.QUALITY}=\mathit{LOCATION.AVG\text{-}QUAL}} \text{LOCATION}$$

  Maybe not sensible, but still doable

  WINE.QUALITY and *LOCATION.AV-QUAL* have the same domain
- There are other join conditions and forms of join (later)

- Projection:

WINE	W#	GRAPE	VINTAGE	PERCENTAGE	QUALITY
	100	Volnay	1979	12.7	Good
	110	Chablis	1980	11.8	Average
	120	Tokay	1981	12.1	Excellent
	130	Chenas	1979	12.0	Good
	140	Volnay	1980	11.9	Average

$\Pi_{\text{VINTAGE,QUALITY}}$

YEAR	VINTAGE	QUALITY
	1979	Good
	1980	Average
	1981	Excellent

- A unary operator
- No duplicates (sets do not have duplicate elements)
- A tuple $\mathbf{t}$ is in the result (the projection) iff there is a tuple $\mathbf{t}'$ in the original relation that, restricted to the attributes indicated in Π gives $\mathbf{t}$: $\mathbf{t}'[\text{VINTAGE}, \text{QUALITY}] = \mathbf{t}$
- For example, $(1979, \mathit{Good}) \in \Pi_{\text{VINTAGE,QUALITY}}(\mathit{WINE})$

  because there there exist values, say x, y, z for attributes $W\#, \mathit{GRAPE}, \mathit{PERCENTAGE}$ (which we do not care about), such that tuple $(x, y, 1979, z, \mathit{Good})$ belongs to relation WINE

- Selection: $\sigma_{<condition>}$

WINE	W#	GRAPE	VINTAGE	PERCENTAGE	QUALITY
	100	Volnay	1979	12.7	Good
	110	Chablis	1980	11.8	Average
	120	Tokay	1981	12.1	Excellent
	130	Chenas	1979	12.0	Good
	140	Volnay	1980	11.9	Average

$\sigma_{QUALITY=Good}$

GOOD-WINE	W#	GRAPE	VINTAGE	PERCENTAGE	QUALITY
	100	Volnay	1979	12.7	Good
	130	Chenas	1979	12.0	Good

- Original attributes are kept

  Here the condition is very simple

- It is possible to express more complex selection (and join) conditions with a language that involves
  - Attribute names
  - Logical, boolean (propositional) operations (AND, OR, NOT)
  - Built-in relations ($=, <, \leq, >, \geq, \neq$) applied to attribute names and domain elements

    E.g. above: $WINE.GRAPE = LOCATION.GRAPE$;
    $QUALITY = Good$

- Built-in relations have a fixed semantics, and fixed and possibly infinite extensions
- In contrast to relation predicates in the schema that have variable extensions depending on the application and the state of the DB
- E.g. the $<$ built-in relation on the data type *integer* has an infinite, fixed extension that the DBMS can simply use
- For example:

$<$	Smaller	Bigger
	0	1
	0	2
	...	...
	1000	1500
	...	...

$\neq$	String	String
	john	peter
	peter	mary
	...	...
	mary	john
	...	...

- A selection could be: $\sigma_{VINTAGE>1980 \ OR \ QUALITY=Good}(WINE)$
- This boolean language can be used to express more complex join conditions: $\bowtie_{<condition>}$

  $WINE \bowtie_{W.GRAPE=L.GRAPE \ AND \ W.QUALITY=L.AVG\text{-}QUALITY} LOCATION$