

Databases

Daniel F. Simola

18 May 2005

1 Problems motivating databases

A database is just a collection of related data. Often times a structured text file will be able to store the entirety of this data set, there are reasons why this might not be the case:

1. Data organization: data structure more complicated than a table
2. Efficiency: more data than RAM
3. Concurrency and Reliability: simultaneous read/write. Concurrency deals with transactions (actions on database items) and recoverability (keeping logs).

2 Database architecture

The architecture of a particular implementation is described using a three-tier fashion:

1. A set of views (projections) from the database (what users see)
2. A conceptual schema: the layout of the entire database (what designers see)
3. Physical level: the file organization structures, algorithm, indexing, etc. that realizes a database
4. Exception! Middleware: If a database becomes too big to query efficiently, another layer is needed to further partition the schema.

3 Database design

The 6-step program:

1. Requirements Analysis: what data, apps, critical operations
2. Conceptual model: High-level description of data and constraints (use entity-relationship (ER) model)
3. Logical conversion into a schema
4. Normalize the data into relations (tables) - relational db model (remove redundancies)
 - table(attribute:domain)
 - domains are like data types (int, string, byte, date, etc.)
5. Physical design: consider workloads, indexes
6. Write applications, add security features

3.1 ER Relationships

- Entities are depictions of tables, with attributes and keys noted.
- Relationships are edges connecting entities as well as the relations they entail.
 - Binary relationships can be classified as 1-1, 1-many, many-many, and different styled edges indicated these types.
 - Edges can also have roles and describe subclass relationships

3.2 Constraints

Constraints limit the freedom of a designer, but allow for numerous mental and computational efficiencies:

- Schemas: structural constraints
- Domains: tabular constraints
 - Relational DBs have limited domains: tables or scalar attributes
 - OO, O-relational DBs allow complex, user-defined domains (lists, classes, etc.)
 - XML DBs allow XML trees and lists of trees
- Keys: a unique subset of attributes that uniquely identifies a tuple.
 - In the case that multiple subsets exist, one is chosen as the ‘primary’ key.
 - If an attribute of one relation refers to a tuple in another relation, that tuple is called a ‘foreign’ key, and must exist for the relation to exist.

3.3 Redundancy and normalization

There are many good and bad ways to construct an entity and a schema. Typically you want to balance schema complexity with efficiency by designing tables as independent as possible. A functional dependency exists when some attribute set determines another attribute set; these dependencies are independent of schema designs. To design a good schema you want all the attributes of a tuple to be determined by the key attributes, ie you don’t want redundancy. To do this you use a Lossless Join Decomposition. An attribute vector R_1, \dots, R_k is a LJD of R_1, \dots, R_n wrt a functional dependency F if every k-tuple satisfies F . Unfortunately you cannot always perform LJDs on tables without keeping some redundancy, so there is a structure called ‘normal form’, of which there are two kinds:

- Boyce-Codd normal form (BCNF): for every relation scheme R and $X \rightarrow A$ over R , either $A \in X$ or X is a superkey for R
- Third normal form (3NF): for every relation scheme R and $X \rightarrow A$ over R , either $A \in X$, X is a superkey for R , or A is a member of some key for R

BCNF is preferable (strictly stronger than 3NF), but can result in ruined dependencies. BCNF is also nondeterministic. On the other hand you can always use 3NF to get lossless joins and dependency preservation.

3.4 XML

XML is the result of a few issues:

- want more declarative queries
- need more flexible data interchange format
- always parsable even if attributes are unknown

. XML allows you to combine almost all techniques, like relational operations, OO and structured data. XML combines schema and data into a single format. The data model looks like

- Document
- Element
- Attribute
- Processing instruction
- Text (content)
- Namespace: provide context for attributes
- Comment

. XML is typically too abstract, eg how do you query? So something called a document type definition (DTD) is also created which defines the structure/grammar of the XML file, namely IDs (aka keys), IDREFs (references to keys), and lists of IDREFs. Even this is not all-encompassing; XML+DTD still cannot capture the idea of domains, IDs are not good replacements for keys, you can't define an inheritance structure, and you can't determine functional dependencies. So something all-powerful was created called the XML Schema, featuring syntax, defining keys, subclassing, built-in datatypes.

4 Relations

A relational algebra contains functions which operate on relations and return relations: $f : R \rightarrow R$: {Projection, Selection, Union, Product, Difference, (Rename), Join, Semijoin, Intersection, Division}. A relational calculus (FOL) can also be used as a query language, and can operate on tuples and domains: $\{x_1, \dots, x_n \mid p\}$, p is a boolean predicate on the variables, which are either domains or tuples.

A few points:

- Relational algebra is procedural/imperative (how?): similar to SQL implementation
- Relational calculus is declarative (what?): similar to SQL language
- the algebra can be mapped uniquely to a domain relational calculus (DRC) and back
- $Product(R_1, R_2)$ combines each tuple in R_1 with each tuple in $R_2 \implies$ huge query. Usually you can just Join, but sometimes you need a Left Outer Join, if an attribute of the Join is null
- There are several operators which have to deal with conflicting name spaces. Need to determine how to deal with this situation.
- Relational algebra has laws of associativity, commutativity, etc, which imply the syntactic equivalence of some relations. Then a SQL optimizer can choose the most efficient option.
- There are issues of safety when executing certain queries, eg the result may be a theoretically infinite set, but this must be prevented.

Limitations:

- Aggregation operations
- Recursive queries
- Complex (non-tabular) relations

These things can typically be done in SQL, OQL, XQL using special operators (more powerful language.)

4.1 SQL language

SQL = structured query language, based on the principles of first-order logic. SQL is the standard language for relational database queries. Its implementation is divided into the data manipulation language (queries) and the data definition language (domains). eg:

```
SELECT c.id
FROM student s, course c
WHERE s.sid = c.sid AND s.name = "JILL"
```

You can also perform computations within queries and operate on their results, perform nested queries, aggregate (GROUP BY: AVG, COUNT, SUM, MAX, MIN, DISTINCT). eg:

```
SELECT subj, AVG(size)
FROM(
    SELECT C.cid AS id, C.subj AS subj,
           COUNT(S.sid) AS size)
FROM STUDENT S, Takes T, COURSE C
WHERE S.sid = T.sid AND T.cid = C.cid
GROUP BY cid, subj)
GROUP BY subj
```