SOFTENG351 Notes 2017

Theodore Oswandi

June 27, 2017

1 Fundamentals of Database Systems

1.1 General Information

Database large integrated collection of data.
Contains [Entities, Relationships]

DBMS (Database Management System):
software package to store and manage databases

Database System : DBMS with database

DBMS and uses

- store large amounts of information
- code for queries
- protect from inconsistencies and crashes
- security
- concurrent access

1.2 Why Databases

Need to shift from computation to storage of large amounts of information

Accomodate for changes in:
Variety: types of data
Velocity: movement of data
Veracity: uncertainty of data
Volume: amount of data

Structures/Models Need to have a model to describe data, and a schema used to give an abstract description of the data model

1.3 Levels of Abstraction

Views: describe how data seen
Logical Schema: how data structures organised (variable types)
Physical Schema: how files structured
Data Definition Language: How to define database schema
Data Manipulation: how to update values in database
Query Language: used to access data

1.4 Data Independence

Logical Data Independence

- external handling separate from logical organisation
- mappings change, not external schema
- applications only see external schema

Physical Data Independence

- changes to physical schema doesn't affect logical layer
- abstract from DBMS storage organisation
- can perform optimisation/tuning

1.5 Concurrency Control

- many users have to be able to access information at the smae time and make updates without negatively affecting database
- don't want to access disk lots. It is slow and inefficient
- let multiple users access and keep data consistent
- let users feel like they're the only ones using system

2 Relational Model of Data

2.1 General Information

- is logical model of data
- distinguish between data syntax and semantics
- simple and powerful
- sql based off this

2.2 Simple approach

- use tuples to store data
- relations are sets of these tuples
- tables to represent sets of data
- properties (columns) are called attributes
- attributes associated with domains (variable types)

2.3 Relational Schemata

Use of attributes creates relation schema such as:
MOVIE(title: *string*, production_year: *number*)

Relation Schema provide abstract description of tuples in relation

Database Schema is set S of relational schemata. Is basically the set of all tables and their attributes

2.4 Keys

Are used to uniquely identify tuples over all data in a given table.
They are used to restrict number of database instances, to something more realistic and identify objects efficiently

Superkey over relation schema is a subset of attributes that satisfies this uniqueness property
Key is a minimal superkey, is key if no other superkeys exist for R

Foreign Key: is a key used to index values from other values. Used to make reference between relational schemata.

- ensures referential integrity
- no need to copy info from other tables
- need to ensure that [x,y] $\subseteq$ [x,y] and not [y,x] (Order matters)

Example
MOVIE(title: string, production_year: number, director_id: number)
with key [title, production_year]

DIRECTOR(id: number, name: string)
with key [id]

with foreign key: MOVIE[director_id] $\subseteq$ DIRECTOR[id]

2.5 Integrity Constraints

- Db schema should be meaningful, and satisfy all constraints
- should stay true to keys, and foreign keys
- constraints should interact with each other correctly
- should process queries and update efficiently
- should do this and make as few comprimises as possible

3 SQL as Data Definition and Manipulation Language

3.1 General Information

- **S**tructured **E**nglish **QUE**ry **L**anguage
- used as a standardised language in relational Db systems
- is query, data definition, and data management language
- names not case sensitive

3.2 Keywords

CREATE used to create tables. [CREATE TABLE $< tablename >< attributespecs >$]

Attributes : defined as [$< attribute\_name >< domain >$]

NOT NULL : used to ensure that attribute is not null.

DROP TABLE : removes relation schemata from Db system

ALTER TABLE : change existing relation schemata

CHARACTER and **VARCHAR** are fixed or variable length strings. Variable length string are $VARCHAR(32)$

NATIONAL CHARACTER : string characters from other languages

INTEGER and **SMALLINT** used for integers

NUMERIC and **DECIMAL** used for fixed point numbers

FLOAT, REAL and **DOUBLE PRECISION** used for floating point numbers

DATE used for dates (year, month, day)

TIME used for times (hour, minute, second)

3.3 Null and Duplicate Tuples

NULL used to insert tuple where some of the information is not known. This is not good as it can create inconsistencies in information. SQL does this through use of a null marker

Duplicate Tuples are when another tuple exist with same values. NULL can make this confusing.
- these duplicated really expensive to remove
- relations do not contain these tuples

3.4 SQL Semantics

Table Schema set of attributes

Table over R set of partial tuples

These relations are idealised with no duplicates

3.5 Constraints

- can add uniqueness parameters like primary keys on table
- NOT NULL makes sure that partial tuples don't exist
- **UNIQUE** $< attribute\_list >$ ensures that values for attributes exist only once
- **PRIMARY KEY** used on tables that are also UNIQUE, and ensures that NOT NULL applied to it too
- $FOREIGNKEY < attributelist > REFERENCES < tablename >< attributelist >$ used to create foreign key dependencies
- $CONSTRAINT < name >< constraint >$ used to give names to constraints

3.6 Referential Actions

Are used to specify what happens if tuples updated or deleted

SQL provies:

- **CASCADE** forces refercing tuples to be deleted
- **SET NULL** sets referencing tuples to NULL
- **SET DEFAULT** sets the tuple values to specified default
- **NO ACTION** does nothing

Example usage: FOREIGN KEY(title, year) REFERENCES MOVIE(title, year) ON DELETE **CASCADE** will delete director tuple if movie deleted

FOREIGN KEY(title, year) REFERENCES MOVIE(title, year) ON DELETE **SET NULL**
will keep tuple in director but won't know what he directed if movie deleted

3.7 CREATE statement

- **CREATE ASSERTION** $< name >$ CHECK defines integrity constraints
- **CREATE DOMAIN** define domains and check clause
- **CREATE VIEW** $< name >$ **AS** $< query >$ define views

- **CREATE GLOBAL TEMPORARY TABLE** makes a table for current SQL session
- **CREATE LOCAL TEMPORARY TABLE** makes table for current module or session

3.8 Use as Data Manipulation Language

Allow insert, delete or update

- INSERT INTO $< tablename >$ VALUES $< tuple >$
- INSERT INTO $< tablename >< attributelist >$ VALUES $< tuples >$
- INSERT INTO $< tablename >< query >$
- DELETE FROM $< tablename >$ WHERE $< condition >$
- UPDATE $< tablename >$ SET $< value - assignment - list >$ WHERE $< conditions >$

4 Relational Query Languages: Algebra

4.1 Query Languages

- allow access and retrieval of data
- foundation based on logic, and allows for optimisation
- are not programming languages
- not intended for complex claculations
- easy efficient access to lots of data

Query Language must have

- formal language with
- input schema
- output schema
- query mapping

Equivalent Queries

- same input schema
- same output schema
- same query mapping
- if L $\sqsubseteq$ L' and L' $\sqsubseteq$ L

Dominant, if Q1 dominates Q2

- if L $\sqsubseteq$ L'

4.2 Relational Algebra

- useful for representing execution plan
- easily translated to SQL
- A is set of possible relations
- Set of Operations:
  - Attribute Selection σ
    returns value where A in MOVIE is same as B in MOVIE
    $[\sigma_{A=B}(MOVIE)]$
  - Constant Selection σ
    returns vlaue where A is 3 in MOVIE
    $[\sigma_{A=3}(MOVIE)]$
  - Projection π
    returns title and date of MOVIE
    $[\pi_{title,date}(MOVIE)]$

  - Renaming $\delta \mapsto$
    renames attribute in MOVIE
    $[\delta_{title \mapsto title'}(MOVIE)]$
  - Union $\bigcup$
    takes 2 arguments and returns set of tuples contained in either result, removing duplicates
    $[\sigma_{A=2}(MOVIE) \bigcup \sigma_{A=3}(MOVIE)]$
  - Difference $-$
    returns negation
    $[-\sigma_{A=3}(MOVIE)]$
  - Natural Join $\bowtie$
    Joins two tables and return tuples that match in both tables
    $[MOVIE \bowtie DIRECTOR]$

Redundant Operations

- Intersection $\bigcap$
  because can use DeMorgans to produce with $-and \bigcup$
- Cross Product $\times$
  because creates $N \times M$ tuples which is not space efficient
- Division $\div$
  because it uses cross product to generate all those that match

$\pi \sigma \bowtie \delta \bigcap \bigcup \bigvee \bigwedge \div \times - \in \mapsto$

4.3 Query Language $\mathcal{L}_{ALG}$ of Relational Algebra

wtf is this

4.4 Incremental Query Formulation

Can use multiple Queries to break up calculation to make it more readable. Ths will also reduce the amount of copying needed as you can use previous results in subsequent calculations.

5 Relational Query Language: Calculus

5.1 General

- let users describe what they want
- simple to write query and translate to SQL
- sometimes relational algebra can get convulated
- based on first order logic
- if safe then can automatically be transformed to SQL style
- **Tuple Relational Calculus TRC** variables represent tuples
- **Domain Relational Calculus DRC** variables represent values in domain

Query Structure & Example
$\{\ to\_return \ \mid \ \exists x, y, z(sometable(x, y, z, n)\}$
$\{x_{name} | \exists x_{age}(PERSON(x_{name}, x_{age}, 'Male'))\}$

5.2 Object Properties

have variables or may set its value in query variable values must be in its domain can make complex properties by using negation, conjunction and exisential quantification remove brackets if doesn't increase ambiguity

5.3 Shortcuts

- **Inequation** $A \neq B$ equivalent to $\neg A = B$
- **Disjunction** DeMorgan's: $A \vee B$ shortcuts $\neg(\neg A \wedge B)$
- **Universal Quantification** DeMorgan's: $\forall x(A)$ shortcuts $\neg\exists x(\neg A)$
- **Implication** $A \Rightarrow B$ shortcuts $\neg A \vee B$
- **Equivalence** $A \Leftrightarrow B$ shortcuts $(A \Rightarrow B) \wedge (B \Rightarrow A)$
- **Successive Exestential Quantification** $\exists x_1(\exists x_2(A))$ same as $\exists x_1, x_2(A)$
- **Successive Univeral Quantification** $\forall x_1(\forall x_2(A))$ same as $\forall x_1, x_2(A)$

5.4 Free variables

- placeholders used to describe structure
- basically doesn't have to hold a specific set of values
- not bound to any quantifier
- negation does nothing to free variables
- obeys conjunction

5.5 Formulae interpretation

Whole thing either evaluates to T or F

5.6 Domain Relational Calculus

- Truth value of formula based on its values on free variables
- **Example:** $\{(m)|\exists d, t(SCREENING('Event', m, d, t))\}$
- if looking for empty set, then just returns if value exists for that or not

5.7 Tuple Relational Calculus

- named on global perspective
- sort logic based on domain and relation schemata
- formula built fromatms on DRC. Free variables defined analogously

5.8 Safe Range Normal Form

- check if query is safe or not
- basically don't want potentially infinite number of values
- Must contain these properties:
  1. **eliminate shortcuts** (Universal Quantification $\forall$, Implication $\Rightarrow$ and Equivalence $\Leftrightarrow$)
  2. **bounded renaming**: ensure no free and bound variables.
     Not allowed: $\neg(A \vee B)$
  3. remove any double negations $\neg\neg A$ to A
  4. only put negation in front of atom or quantifier, and not over bracketed area
  5. omit unecessary brackets
  6. ensure there is no disjunction in final formula
- if all free variables range restricted then formula safe
- if in SRNF then everything should either be an atom or quantified formula

6 SQL: As a Query Language

Basic structure:
$SELECT < attributelist > FROM < tablelist > WHERE < condition >$

Renaming (tables and attributes):
$< name > AS < newname >$

Select and remove duplicates
$SELECTDISTINCT \ldots$

Sort results (can sort by multiple in heirarchy):
$ORDERBY < attributelist >$
add ASC or $DESC$ to determine if want increasing/decreasing

6.1 Operators:

$=, <, >, \leq, \geq, <>$(not equal to)

6.2 Where Structure

$[< expression >< operator >< expression >]$

6.3 Values in a range

$[< attribute > BETWEEN < value > AND < value >]$
$[< attribute > IN < valuelist >]$

6.4 Aggregate functions

- **COUNT**: returns number of values
- **AVG**: return average argument values. Only work if domain some kind of number
- **MIN**: returns minimum argument value
- **MAX**: returns maximum argument value
- **SUM**: returns sum of argumet values. Only works if domain is number.

Results of above functions different if distinct used

6.5 Grouping

- used when only want to apply aggregate on group of tuples
- use $GROUPBY < attributelist >$
- if want to only do some groups then add $HAVING < condition >$
- the HAVING condition must apply to group and not tuples in groups

6.6 Subqueries

- can utilise further queries in the WHERE conditions
- **IN** $< subquery >$ checks if tuple exists in subquery
- **EXISTS** $< subquery >$ checks if subquery returns non empty relation
- **UNIQUE** $< subquery >$ checks if result contains no duplicates
- can **NOT** above queries if you want

6.7 Reachability

- basically seeing if given a graph if you can go from A to X in n number of stops
- If n is known then can use iteration
- However if n is flexible then must use recursion somehow

6.8 Recursion

wat.

6.9 Writing Difficult SQL Query

1. Formalise query in safe calculus
2. Transform query into SRNF
3. Transform SRNF to SQL

6.10 Division

Basically if $R \div S$, then return all attributes in R that satisfy all values in S, except columns in S. utilises 2 nested WHERE NOT EXISTS

6.11 Outer Joins

- used when $t_1 \in R_1$ don't match $t_2 \in R_2$. So can't just do $R_1 \bowtie R_2$
- **FULL OUTER JOIN**
  if r FULL OUTER JOIN s then do join, keep all information from both tables and make unknown attributes NULL
- **LEFT OUTER JOIN**
  if r LEFT OUTER JOIN s, then do the join, and keep entirety of r, those values not in s make resulting column for that tuple NULL
- **RIGHT OUTER JOIN**
  if r RIGHT OUTER JOIN s, then do join and keep entirety of s but only match r if exists. Unknown values become NULL

Possibly make list of examples here

7 Entitiy Relationship Model

7.1 Database Design

7.1.1 General Information

- Organisations now have access to lots of data
- Data is recorded in databases
  - What to keep in Db?
  - How to access it?
- Database design aims to create and manage complex databases for huge information systems
- Broken down into four phases
  1. Requirement Analysis
  2. Conceptual Design
  3. Logical Design
  4. Physical Design

7.1.2 Conceptual Design

target of database meet organisation that is going to use database needs

conceptual design provides abstract description of database in terms of high level concepts. Shows expected structure

Inputs are information requirements of users

Output is database schema, with consideration of user requirements but no layout, implementatation and phyiscal details yet

Conceptual data model to describe language

7.2 Entity Relationship

7.2.1 Entities

- basic objects
- described by attributes
- these attributes should allow differentiation between objects
- **key** created from these attributes to uniquely identify objects

Look

- Visualised by rectangles
- Attributes point to rectangle
- Attriutes that create key are underlined

7.2.2 Relationships

- create connection between various entities
- are also technicalle objects that connect other objects together

Format: COMPONENTS, ATTRIBUTES, KEYS
Examples: $\{\{Student : PERSON\}, \{Year, Course\}, \{Student, Year, Course\}\}$
Look

- Visulaised by diamonds
- Contains attributes as well, but doesn't have to.
- Same as Entities, key attributes are underlined
- Edged linked to it are objects it associates with
- Components that are part of key have a dot in the line connecting it to relationship diamond
- key may span all attributes

Roles

- used to tie together relationship containing at least 2 of hte same object type. Both will have a line connecting diamond and rectangle, each named respectively.
- effectively make use of foreign keys

Example: $\{\{Student : PERSON, Lecturer : PERSON\}, \emptyset, \{Student, Lecturer\}\}$

unique-key-value property exists due to attributes in object uniquely identifying tuples. In a given set of tuples no duplicates in attribute subset that makes up key will exist.

7.3 Set vs Foreign Key Semantics

Set semantics uses entire attribute set
Foreign key semantics uses only key attributes

If using foreign key semantics then definitions of [Relationships, Relationship Sets, Database Instances] are slightly different. But also lets you use E_{id} instead of having to use entire E to reference things.

Due to unique-key-value property, using either E or E_{id} is equivalent since can identify tuple by key attributes

7.4 Identifiers

For given Entity, can create new attribute that identifies rest of attributes for given tuple. If can ensure this column is unique then can use it as single attribute key. This kind of associates an attribute with an entity

unique identifier property bascally saying that identifiers that identify relationships must be unique

7.5 Specialisation, Generalisation, Clusters

7.5.1 Specialisation

- lets you define multiple types of an entity. Basically subtypes
- Such as Student, Lecturer, Tutor which are all people
- Generally adds various attributes that relate to specialised object type.
- Is treated as a relation
- Can have specialisations of specialisations (relationship of relationship)

Subtype U with supertype C: $U = (\{C\}, attr(U), \{C\}$

7.5.2 Generalisation

- abstract concepts that model multiple things
- allows you to create a single relationship for connections

7.5.3 Clusters

- model disjoint union (or)
- ensure that things are mutually disjoint
- attach those cluster groups to a single point $\oplus$
- cluster still labelled as an object

7.6 Transforming the ER diagram

- these diagrams can be transformed to database schemata for things such as importing into SQl for example.
- Entities become tables
- Relationships become tables with foreign key connections to other tables

7.7 Handling Clusters

- clusters in conceptual design to model alternatives
- to handle them, must remove them from ER diagram to see individual interactions.
- cluster provide convenient way to model objects to Db
- if you avoid clusters then ER schemata becomes large

8 Database Design Quality: Relation Normalisation

8.1 General

- **Normalisation**'s goal is to efficiently process updates
  - this aims to have no data redundancy
  - redundancy means you have to do many updates
  - also means you have to keep checking that data integrity exists and all redundant values updated as needed.
- **Denormalisation**'s goal is to efficiently process queries

  - because joins are expensive to do
- cannot always do this as they trade off each other
- therefore try to design datbase well

8.2 Functional Dependencies

- if $X \to Y$ then for a given value of X, the Y value will be same.
- therefore if $Student \to Teacher$ then that student will always have that teacher
- if $\emptyset \to X$ then X can only have one value
- attributes occuring in a key is a **prime attribute**

8.3 Derivation Rules

General formula $\frac{IF}{THEN}$

Transitivity $\frac{X \to Y, Y \to Z}{X \to Z}$

Extension $\frac{X \to Y}{X \to XY}$

Reflexivity $\frac{}{XY \to Y}$

Also something about derivation trees but I don't understand

8.4 Soundness & Completeness

Sound R is sound if every derivable dependency is implied

Complete R is complete if every implied dependency is derivable

8.5 Canonical Cover

Get all functional dependencies and create a minimal list that still preserves its properties

Contains:

1. Get all Functional Dependencies
2. **Decompose**: break $A \to BC$ to $A \to B, \to C$
3. **L-reduced cover** which is when you do $AB \to C$ to $A \to C$
4. Then you are left with canonical cover when you delete duplicates and redundant dependencies.

8.6 Boyce-Codd Normal Form

- can always get lossless BCNF decomposition
- guarantee no redundancy in terms of FD
- might exist some FD that can't be enfored
- no partial key can depend on superkey

8.7 Third Normal Form

- Lossless 3NF can always be achieved
- guarantees that all FDs can be enforced on elements in 3NF decomposition
- ensures least level of data redundancy among all lossless, faithful decompositions

9 Gerald's Fun Section