Fiche Database

Pierre Colson

Contents

Markdown version on *github*
Compiled using *pandoc* and *`gpdf` script*
More fiches *here*

General

- A database should allow the user to specify thery schemas usiong a **data definition language**, and allow to query and modify data using a **query/data manipualtion language**
- Database transactions should respect **ACID**
  - **A**tomicity (all or nothing)
  - **C**onsistentcy (regarding constraints & transcations preserve them)
  - **I**solation (apper to be executed as if by itself)
  - **D**urability (transacations nevet lost onece competed)
- A **Data Model** consists of :
  - A mathematical representation if data
  - Operation on data
  - Constraints
- **Attributes** is the column names
- **Component** is an instance of an attribute (a column cell)
- **tuple** is a row

- A **key** is a sets of attributes that determine all other attributes if a tuple

Algebra

- **Relational Algebra**

  Ordered by Precedence

  - σ_c Selection
  - π_L Projection
  - ρ Renamming
  -
  - $\times$ Cartesian Product
  - $\bowtie_c$ Theta Join ($R \bowtie_c S = \sigma_c(R \times S)$)
  - $\bowtie$ Natural Join ($R \bowtie S = \pi_L(\sigma_c(R \times S))$)
  -
  - $\cup$ Union
  - $\cap$ Intersection
  - $-$ Difference

- **Extended Relational Algebra**

  - δ Duplicate Elimination
  - τ Sorting
  - γ Grouping and Aggrgation
  - π_L Extended Projection
  - `correct symbol` Outer Join, Left Outer Join, Right Outer Join

- A relation under **bad semantics** contains a bag of tuples (allow duplicate)

- A relation under **set semantics** contains a set of tuples (no duplicate)

- **NULL** Values

  Can be used for:

  - Value Unknown
  - Value Inapplicable
  - Value Witheld

  Other:

  - Does not contributes to `SUM/AVG` but contributes to `COUNT`
  - True is 1, False is 0 and Unknown is 0.5
  - A AND $B = min(A, B)$
  - A OR $B = max(A, B)$
  - NOT $A = 1 - A$

SQL

- Create a realtion

```
CREATE TABLE Name(
    attribute1 type,
    attribute2 type
);
```

- Delete a relation

```
DROP TABLE Name
```

- Declare primary key

```
CREATE TABLE Name(
    attribute1 type primary key,
    attribute2 type
);
```

```
CREATE TABLE Name(
    attribute1 type,
    attribute2 type
    primary key(attribute1, attribute2)
);
```

- Unique COnstraints
- Declare primary key

```
CREATE TABLE Name(
    attribute1 type unique,
    attribute2 type
);
```

```
CREATE TABLE Name(
    attribute1 type,
    attribute2 type
    unique(attribute1, attribute2)
);
```

Key Differences Between Primary key and Unique key:

- Primary key will not accept NULL values whereas Unique key can accept one NULL value.
- A table can have only primary key whereas there can be multiple unique key on a table.
- A Clustered index automatically created when a primary key is defined whereas Unique key generates the non-clustered index.

- Foreign Keys

```
CREATE TABLE name(
    attribute1 type,
    attribute2 type,
    attribute3 type,
    FOREIGN KEY (attribute1) REERENCES otherTableName(attribute1)
    FOREIGN KEY (attribute2) REERENCES otherTableName(attribute2)
)
```

- Insertion

```
INSERT INTO tableName VALUES(component1, component2, etc)
```

- Basic Queries

```
SELECT attribute1, attribute2 AS otherAttributeNAme
FROM tableName
WHERE condition;
```

```
...
WHERE LIKE 'Ma%' OR LIKE 'M_ke'
```

- Semantics

- For Union, Intersection or Difference by default its set semantics
    - Under set semantics
      ```
      (SELECT ... FROM ... WHERE)
      UNION
      (SELECT ... FROM ... WHERE)
      ```
    - Under bag semantics
      ```
      (SELECT ... FROM ... WHERE)
      UNION ALL
      (SELECT ... FROM ... WHERE)
      ```
- For selections bag semantics is by default
    - Select under ser semantics
      ```
      SELECT DISTINCT ... FROM ... WHERE
      ```

- Subqueries

```
SELECT attribute1
FROM
    (SELECT ...
    FROM ...
    WHERE ...) A
WHERE A.attribute = 'smthg'
```

- `EXISTS(sbuquery)` evaluates to true if subquery is not empty

Have a good database

- K is a **superkey** for a realation if K functionally determines all of R
- K is a **key** for R if K is a superkey but no proper subset of K is a superkey
- A functional dependency $X \to Y$ such that Y is a subset of X is called **trivial**.
- Let S be a set in functional dependencies for R and let X be a set of attributes of R. The **closure** of X denoted X^+ is the largest set of attributes such that every tuple in R that satisfies all functional dependencies in S also satisfies $X \to X^+$
- To find keys and superkeys you have to compute all closure of all subsets of attributes
- Suppose some set of FD's S holds a relation R. Any set of equivalent FD's is called a basis for R
- A **minimal Basis** for a relation is a basis B such that:
    - All the FD's have singleton right hand side
    - If any FD is removed form B, the result is no longer a basis
    - If for any FD F in B we remove one or more attributes from the left hand side of F, the result is no longer a basis
- The projection $\pi_L(S)$ of FD S under π_L is the set of FD's that:
    - Follow from S
    - Involve only attributes of $\pi_L(R) = R_1$
- Goal of Decomposition :
    - NO anomalies
    - Lossless Join: we can recover R with a natural join $R = R_1 \bowtie \ldots \bowtie R_n$
    - Dependency preervations

- The **Chase Test** allow us to chekc if a decompostion is lossless or not:
  - Apply FD's to tableau whenever possible to remove "subscripts/vairables"
    - If we arrrive aa completely subscript free row: lossless
    - If not, resulting tuples $\{t_1, \ldots, t_n\}$ porvide an instance of R with a concrete counter example.
- Boyce Codd Normal Form **BCNF**, we say that a relation is Byyce-Codd Normal form (BCNF) if whenever $X \to Y$ is a nontrivial FD that holds in R then X is a superkey. (The left hand side of any nontrivial FD's lust contain a key)
- BCNF Algorithm

  Input (R, F): Relation R and FD F

  - Search for a BCNF violation $X \to Y$ in F, is there are none, R is in BCNF
  - Replcae R by $R_1 = X^+, R_2 = R - (X^+ - X)$
  - Compute projections of FD's F_1 and F_2 of the FD F onto R_1 and R_2 repectively.
  - Apply the algorithm to (R_1, F_1) and (R_1, F_2), return the union of the decomposition as result.

    Goal:
  - BCNF avoid anonmalies
  - The result is lossless
  - Result may not be dependency preserving
- A **prime attribute** if this attribute is a menber of any key.
- Third Normal Form **3NF**
  - We say nontrivial $X \to A$ violates 3NF if and only if X is not a superkey and A is not prime.

    Algo : Input: Relation R and FD F
  - Find a minimal basis for F, say G
  - For each FD $X \to A$ in G use XA as the schema of one of the relations in the decomposition.
  - If one of the relations in step 2 is a superkey for R, add another relation whose attributes are a key for R

    Goal:
  - 3NF does not always avoid anomalies
  - 3NF is lossless
  - 3NF is dependency preserving

Concurrency and Transactions

- **ACID** transactions enforce
  - **A**tomic: Whole transcation or nothing is executed
  - **C**onsistent: Datatbase constraints are preserved
  - **I**solated: It apperas to user as if no other transaction happening concurrenlty
  - **D**urable: Robust under errors/ crash
- Isolation Level:

  Transactions:

  - Serializable: Transactions sees committed data state as it is at the beginning of transaction. Only modifications done by the transactions itself are visible from within transaction

    ```
    BEGIN TRANSACTION
    SET ISOLATION LEVEL SERIALIZABLE
    ...
    COMMIT
    ```

  - Read Uncommitted: Transaction may see even changes by other transactions that were not committed yet.

    ```
    BEGIN ...
    SET ISOLATION LEVEL READ UNCOMMITED
    ...
    ```

  - Read Committed: Transaction may be see only committed changes from other transactions (however, what is committed by others's may cahnge during execution of our transaction)

    ```
    SET ISOLATION LEVEL READ COMMITED
    ```

  - Repeatble Read: Like Read Committed, but if a row is read agian duraing transaction all original tuples are guaranted to be returned agian (but it may result in more tuples as well)

    ```
    SET ISOLATION LEVEL REPEATABLE
    ```

    Tuples

  - Dristy Read: When data that is not yer committed by transaction 1 is read bu another trasaction 2. This causes problem when transaction 1 is oborted
  - A non-repeatable read occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.
  - A phantom read occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

.	Dirty Reads	Non-repeatable Reads	Phantoms
Read Uncommitted	yes	yes	yes
Read Committed	no	yes	yes
Repeatable Read	no	no	yes
Serializable	no	no	no

Simple example:

- User A runs the same query twice.
- In between, User B runs a transaction and commits.
- Non-repeatable read: The A row that user A has queried has a different value the second time.
- Phantom read: All the rows in the query have the same value before and after, but different rows are being selected (because B has deleted or inserted some). Example: select sum(x) from table; will return a different result even if none of the affected rows themselves have been updated, if rows have been added or deleted.

View, Constraints, Assertions, Trigger

- A **View** is a relation defined in terms of stored tables and toher views
  - Virtual: The view realtion is just a query referencing source realtions, not stored itself
  - Materialized: The view realtion is constructed and explicitely stored

- A **Constraints** Is a relationship among dataelements in DBMS that must hold at all times
- Can add a check on an atrribute, works only on insertion or updated but not for deletion

```
CREATE TABLE tableName (
    attribute1 type CHECK (conditon)
    ....
    CHECK (condition on attributeX and attributeY)
)
```

- **Assertions** check an arbitrary boolean valued SQL condition

```
CREATE ASSERTION assertionName (
    CHECK (condition)
)
```

- **Triger** or Event-Condition-Action

```
CREATE TRIGGER triggerName
    AFTER INSERT IN tableName
    REFERENCES NEW ROW AS newtuple
    FOR EACH ROW
    WHEN (newtuple.attribute NOT IN (SELECT attribute FROM otherClassName))
    INSERT INTO otherClassName VALUES(newtuple.attribute, otherAttribute, etc)
```

- Row level trigger: execute once for each modified tuple
- Statement-level triggers: execute once for an SQL statement

Schema design in the real world

- Entity/Realtion Models, this model allows us to sketch database schema designs
  - Includes some constraints, but not operations
  - Designs are pictures called Entity-Relationship diagrams
  - There is a structured process to turn ER diagrams into realtional database schemas

    ER Diagrams
  - Entity set: represented by a rectangle
  - Attribute: represented by oval with a line to the entity set rectangle
  - The key attribute are underlined, they are required for every entity set
  - Relationsships represent connections between two or more entity sets, they are represented by a diamond
  - Many-one relationship indicated by arrow entering "one" side
  - One-one realtionship indicated by arrows entering both sides
  - If arrow tip rounded: each entity from source is realted to exactly one entity from the target
  - ISA triangles pointing to superclass indicate relationship
  - An entity set E is called weak ti, in order to identity entities of E uniquely
- Subclasses: Three conversion approaches
  - Use null: One relatin entitues have null in attributes that don't belong to them
  - E/R Style: one reation for each subclass with key attributes and attributes of that subclass
  - Object oriented: One relation per subtree of subclasses with all relevant attributes

Semistructured-Data Model

- A database of semistructured data is a collection of nodes. Each node is either interior or a leaf
  - Leaf nodes have associated data of atomic type
  - Interior nodes have a nimber of arcs out
  - Arcs have labels describing how nodes at head and tail of arc are related
- **XML**

  Different way to store data:
  - `<Course code="DD1334"><Capacity>200</Capacity></Course>`
  - `<Course code="DD1334" capacity=200></Course>`
  - `<Course code="DD1334" capacity=200/>`
- DTD file :
  - Therory
    * Subtags must appear in order shown
    * '*' = zero or more times
    * '+' = one or more times
    * ? = zero or once
    * used to indicate "or" alternative tags
    * #IMPLIED = optional attribute
    * #REQUIRED = necessary attribute
    * ID = a name for a unique id
    * IDREF = a reference to an ID taking that value
    * IDREFS = a reference to a set of Ids separated by space
    * CDATA = character string data with special chars escaped
    * <!DOCTYPE yourRootTagName SYSTEM "filename.dtd">

```
<! DOCTYPE root-tag [
    <! ELEMENT element-name (components) >
    <! ATTLIST element-name
        att-name1 type1
        att-name2 type2
    >
    ...more elements
]>
```

  - Full example
    * DTD

```
<!DOCTYPE StarMovieData [
    <!ELEMENT StarMovieData (Star* , Movie*)>
    <!ELEMENT Star (Name, Address+)>
    <!ATTLIST Star
        starld ID #REQUIRED
        starredln IDREFS #IMPLIED
    >
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movie (Title, Year)>
    <!ATTLIST Movie
        movield ID #REQUIRED
        starsOf IDREFS #IMPLIED
    >
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

* XML

```
<StarMovieData>
    <Star starID = "cf" starredIn = "sw">
        <Name> Carrie Fisher </name>
        <Address>
            <Street> 123 Mapple St. </Street>
            <City> Hollywood </City>
        </Adress>
    </Star>
    <Star starId = "mh" starredIn = "sw">
        <Name> Mark Hamill </Name>
        <Address>
            <Street> 456 Oeak Rd. </Street>
            <City> Brentwood </City>
        </Address>
    </Star>
    <Movie movieID = "sw" starsOf = "cf mh">
        <Title> Star Wars </Title>
        <Year> 1977 </Year>
    </Movie>
</StarMovieData>
```

- XMl Schema elements

```
<xs:element name="Name" type="xs:String">
    <xs:key name = "key name">
        <xs:selector xpath = "path">
        <xs:field xpath = "path">
    </xs:key>
    <xs:attribute name = "title" type = "xs:string"
        use = "required"/>
    <xs:attribute name = "year" type = "xs:int"
        use = "required"/>
    <xs:restriction base = "xs:integer">
        <xs:minInclusive value = "1915" />
    </xs:restriction>
    etc
</xs:element>
```

Query Languages for XML : Xpath and Xquery

- XPath
  - Simple Path expressions are sequences of slashes / and tags, starting with /. To construct result, start with doc node and process each tag left to right
  - Indicated with @attributename to get an attribute instead of subelement
  - Instead of searching for matches from root node, we can use //X, then the first step can begin at the root or any subelement of the root as long as the tag is X.

  - We can use wildcard symbol * to match any tag.
  - A condition [...] may follow a tag to only select results that satisfy condition. We refer to current element content with "."

    ```
    /StarMovieData/Movie/Year[.>1976]
    /StarMovieData/Movie[@movieID = "sw"]
    ```

- Xquery
  - Each for creates a loop and let produces a local definition
  - At each iteration of nested loop, if any, evaluate where clause
  - If where clause returns true, or in all cases if no where clause, evaluate return clause
  - For variable in expression variables begin with $
  - Surround variable name by {...} to return value held

    ```
    let $d:=document("file.xml")
    for $data in $d/root/blabla/exmaple
    order by $data descending
    let $temp := (
        for $data in $d/root/blabla/exmaple
        where $data/@id = $blabla
        return $data
    )
    return
        <NewData>
        {$data}
        </NewData>
    ```

- Indices
  - An index on an attribute A is a datastructure making it efficient to search for tuple with a particular value for that attribute. We may also specify indices for multiple attributes at once.
  - This typically comes at the cost of more expensive update/delete/ insert operations.
  - In fact, indices on keys are extremely common as we expect to search by key attributes.

    ```
    CREATE INDEX GradeIndex ON Grades(studentId, courseId)
    ```