Database Systems (Elmari·Navathe) Midterm Review

Best Wishes from pochun

November, 2020

1 Relational Model

Vocabulary:

- **Data model**.

  A notation for describing data.

  A set of operations used to manipulatee that data.

- **Domain**.

  A set of atomic values.

- **Relation**.

  A finite subset of the Cartesian product of a list of domains.

- **Relational schema R**.

  Denoted by $R(A_1, A_2, ..., A_n)$ where R is a relation name.

- **Attribute**

  Each attribute A_i is the name of a role played by some domain D.

  D is called the domain of A_i and is denoted by $dom(A_i)$.

Characteristic of relations:

- No order among tuples.

  No preference for one logical ordering over another.

- Each value in a tuple is an atomic value.

  Composite and multivalued attributes are not allowed.

Relational constraints and relational database schemas:

- **Domain constraints**.

  Each attribute must be an atomic value.

- **Key constraints**.

  All tuples in a relation must be distinct.

- **Candidate key**.

  A relation schema may have more than one key.

- **Primary key**.

  One of the candidate keys.

- **Relational database schema S**

  A set of relation schema $S = \{R_1, R_2, ..., R_m\}$ and a set of **integrity constraints**.

- **Relational database state DB of S**

  A set of relation states $DB = \{r_1, r_2, ...r_m\}$

  Each r_i is a state of R_i and the r_i relation state satisfies the integrity constraints specified in IC.

- **Entity integrity constraint**.

  No primary key value can be NULL.

- **Referential integrity constraint**.

  Maintain the consistency among tuples of the two relations.

  A tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

- **Foreign keys constraint**.

  A set of attributes FK in R_1 is a **foreign key** of R_1 that references R_2 if it satisfies:

  (1) The attributes in FK have the same domain(s) as the primary key attributes PK of R_2.

  (2) A value of FK in a tuple t_1 of $r_1(R_1)$ either occurs as a value of PK for some t_2 of $r_2(R_2)$ or is NULL.

- **Semantic integrity constraints**.

  E.g., the salary of an employee should not exceed the salary of the employee's supervisor.

Operations:

- **Insert**.

- **Delete**.

- **Update**.

- **Select**. $\sigma_{<selection\ condition>}(< relation\ name >)$

  E.g., $\sigma_{(<DNO=4\ OR\ DNO=5>)AND(SALARY>30000)}(EMPLOYEE)$

  Select operation is *commutative*.

  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$

  $\sigma_{<cond1>}(\sigma_{<cond2>}(...\sigma_{<condn>}(R))) = \sigma_{<cond1>AND<cond2>AND...AND<condn>}(R)$

- **Project**. $\pi_{attribute\ list>}(< relation\ name >)$
  $\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$
- **Union Compatible**.
  Two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$ are union compatible if they have the same degree n and if $dom(A_i) = dom(B_i)$ for $1 < i < n$.
- **Union ($R \cup S$)**.
- **Intersection ($R \cap S$)**
  $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$.
- **Difference ($R - S$)**.
- **Cartesian product ($R \times S$)**.
- **Join**.
  Given two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$, $R \bowtie_{<join\ condition>} S = Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_n)$ where the tuples in Q satisfy the join condition.
  $R \bowtie_{<join\ condition>} S = \sigma_{<condition>}(R \times S)$
  Natural join ($*$) requires that the two join attributes have the same name.
- **Complete set of relational algebra**: σ, π, $\cup$, $-$, and $\times$
- **Aggregate functions**: Sum, Average, Maximum, Minimum, and Count.

2 SQL (Structured Query Language)

Data definition in SQL:

- SQL uses *table*, *row*, and *column* for relation, tuple, and attribute, respectively.
- SQL commands for data definition: **CREATE**, **ALTER**, and **DROP**.
- SQL *schema*, identified by a *schema name*, includes an authorization identifier and *descriptors* for each element.
  E.g., **CREATE SCHEMA** *COMPANY* **AUTHORIZATION** JSMITH.
- Data types in SQL: numeric, character-string, bit-string, date, time, and interval.
- **DROP** commands: **DROP SCHEMA**, **DROP TABLE**
- **DROP** behavior options: **CASCADE** or **RESTRICT**.
  RESTRICT options:

A schema is dropped only if it has no elements in it.

A table is dropped only if it is not referenced in any constraints.

- Possible **ALTER** table actions:

  Adding or dropping a column, changing column definition, adding or dropping table constraints.

- SQL allows a table to have two or more tuples that are identical in all their attribute values.

  Hence, SQL table is not a *set* of tuples, rather it is a *multiset* (bag).

SELECT < *attribute list* >
FROM < *table list* >
WHERE < *condition* >

- Aliasing.

  SELECT E.NAME, E.ADDRESS
  FROM EMPLOYEE E, DEPARTMENT D
  WHERE D.NAME = 'Research' **AND** D.DNUMBER = E.DNUMBER

- A missing **WHERE** indicates no condition on tuple selection

- ***** in **SELECT** stands for all the attributes

- **DISTINCT** in **SELECT** can be used to eliminate duplicate tuple

- Set operation: **UNION**, **EXCEPT** , and **INTERSECT**

  Duplicate tuples are eliminated from the result by using those set operations.

- Retrieve all employees whose address is in Taipei, Taiwan.

  SELECT FNAME, LNAME
  FROM EMPLOYEE
  WHERE ADDRESS **LIKE** '%Taipei, Taiwan%'

- **ORDER BY**.

- Set comparisons: **IN** and **CONTAINS**

- **ANY** (= **SOME**) and **ALL**

- Nested queries.

  E.g., Retrieve the name of each employee who works on all the projects controlled by department 5.
  SELECT FNAME, LNAME
  FROM EMPLOYEE
  WHERE
  ((**SELECT** PNO **FROM** WORKS_ON **WHERE** SSN = ESSN)

CONTAINS

(**SELECT** PNUMBER **FROM** PROJECT **WHERE** DNUM = 5))

Similar to ÷ in algebra expression.

- **EXISTS**

  E.g., List the names of managers who have at least one dependent.

  SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE

  EXISTS(**SELECT** * **FROM** DEPENDENT **WHERE** SSN = ESSN)

  AND

  EXISTS(**SELECT** * **FROM** DEPARTMENT **WHERE** SSN = MGRSSN)

  is equivalent to

  SELECT FNAME, LNAME

  FROM EMPLOYEE, DEPARTMENT, DEPENDENT

  WHERE SSN = MGRSSN **AND** MGRSSN = ESSN

- **NOT EXISTS**

  E.g., Retrieve the name of emplyees who have no dependents.

  SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE

  NOT EXISTS (**SELECT** * **FROM** DEPENDENT **WHERE** SSN = ESSN)

- **IN**

  E.g., Retrieve the social security numbers of all emplyees who work on project number 1, 2, or 3.

  SELECT DISTINCT ESSN

  FROM WORKS_ON

  WHERE PNO **IN** (1,2,3)

- **IS NULL**

  E.g., Retrieve the names of all employees who do not have supervisors.

  SELECT DISTINCT FNAME, LNAME

  FROM EMPLOYEE

  WHERE SUPERSSN **IS NULL**

- **AS** (Renaming attributes)

  E.g., Retrieve the last name of each employee and his/her supervisor.

  SELECT E.LNAME **AS** EMPLOYEE_NAME, S.LNAME **AS** SUPERSSN_NAME

FROM EMPLOYEE **AS** E, EMPLOYEE **AS** S
WHERE E.SUPERSSN = S.SSN

- **JOIN + ON** (Joining tables)
  SELECT FNAME, LNAME, ADDRESS
  FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** DNO = DNUMBER)
  WHERE DNAME = 'Research'

- Aggregate functions: **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
  The aggregate functions can be used in **SELECT** and **HAVING**.
  E.g., Retrieve the number of employees in the 'Research' department.
  SELECT COUNT(*)
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO = DNUMBER **AND** DNAME = 'Research'

- **GROUP BY**
  E.g., For each department, retrieve the department number, the number of employees in the department, and their average salary.
  SELECT DNO, **COUNT**(*), **AVG**(SALARY)
  FROM EMPLOYEE
  GROUP BY DNO

- **HAVING**
  E.g., For each project on which more than to employees work, retrieve the project number, the project name, and the number of employees who work on the project.
  SELECT PNUMBER, PNAME, **COUNT**(*)
  FROM PROJECT, WORKS_ON
  GROUP BY PNUMBER
  HAVING COUNT(*) > 2

SELECT < *attribute list* >
FROM < *table list* >
(**WHERE** < *condition* >)
(**GROUP BY** < *grouping attribute* >)
(**HAVING** < *group condition* >)
(**ORDER BY** < *attribute list* >)

- **WITH** statements.
  WITH BIGDEPT (DNO) **AS**
  (**SELECT** DNO

FROM EMPLOYEE

GROUP BY DNO

HAVING COUNT(*) > 5)

SELECT DNO, **COUNT**(*)

FROM EMPLOYEE

WHERE SALARY > 40000 **AND** DNO **IN** BIGDEPT

GROUP BY(*) DNO

is equivalent to

SELECT DNO, **COUNT**(*)

FROM EMPLOYEE

WHERE SALARY > 40000 **AND** DNO **IN**

(**SELECT** DNO

FROM EMPLOYEE

GROUP BY DNO

HAVING COUNT(*) > 5)

GROUP BY(*) DNO

- **RECURSIVE** statements.

  To iterate is human, to recursive, divine!

  E.g., Find all the supervisors of each employee.

  WITH RECURSIVE EMP_SUP (EmpSSN, SuperSSN) **AS**

  (**SELECT** SSN, SUPERSSN

  FROM EMPLOYEE

  UNION

  SELECT E.SSN, S.SupSSN

  FROM EMPLOYEE **AS** E, EMP_SUP **AS** S

  WHERE E.SUPERSSN = S.EmpSSN)

  SELECT *

  FROM EMP_SUP

Update ststements in SQL:

- **INSERT**

  E.g.,

  INSERT INTO EMPLOYEE (FNAME, MINT, LNAME, SSN)

  VALUES ('Lelouch', 'vi', 'Britannia', '123456789')

- **DELETE**

  E.g.,

DELETE FROM EMPLOYEE
WHERE DNO **IN**
(**SELECT** DNUMBER
FROM DEPARTMENT
WHERE DNAME = 'Research')

- **UPDATE**

  E.g.,
  UPDATE EMPLOYEE
  SET SALARY = SALARY * 87
  WHERE DNO **IN**
  (**SELECT** DNUMBER
  FROM DEPARTMENT
  WHERE DNAME = 'Research')

- **CASE** statements.

  UPDATE EMPLOYEE
  SET SALARY =
  CASE
  (**WHEN** DNO = 5 **THEN** SALARY + 1126
  WHEN DNO = 4 **THEN** SALARY + 2252
  WHEN DNO = 3 **THEN** SALARY + 7746
  ELSE SALARY + 0)

View (virtual table): a single table derived from other tables.

No limitation for querying; some limitations for updating.

- **CREATE**

  E.g.,
  CREATE VIEW WORKS_ON1
  AS SELECT FNAME, LNAME, PNAME, HOURS
  FROM EMPLOYEE, PROJECT, WORKS_ON
  WHERE SSN = ESSN **AND** PNO = PNUMBER
  or
  CREATE VIEW DEPT_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
  AS SELECT DNAME, COUNT(*), SUM(SALARY)
  FROM DEPARTMENT, EMPLOYEE
  WHERE DNUMBER = DNO
  GROUP BY DNAME

DROP VIEW WORKS_ON1

DROP VIEW DEPT_INFO

- **UPDATE**

  An update on a view defined on a single table without any aggregate functions can be mapped to an update on the base table.

  UPDATE WORKS_ON1

  SET PNAME = 'ProductPCW'

  WHERE LNAME = 'Smith' **AND** FNAME = 'John' **AND** PNAME = 'ProductX'

- Updatable?

  A view with a single defining table is updatable if attributes contain primary key or other candidate key of the base table.

  Views defined on multiple tables using joins are generally not updatable

  View defined using grouping and aggregate functions are not updatable

- **ASSERTION**

  E.g., The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

  CREATE ASSERTION SALARY_CONSTRAINTS

  CHECK (**NOT EXIST**

  (**SELECT** *

  FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D

  WHERE E.SALARY > M.SALARY **AND** E.DNO = D.DNUMBER **AND** D.MGRSSN = M.SSN))

- **TRIGGER**

  A trigger specifies a condition and an action to be taken in case that condition is satisfied.

  DEFINE TRIGGER SALARY_TRIG **ON** EMPLOYEE E, EMPLOYEE M, DEPARTMENT D:

  E.SALARY > M.SALARY **AND** E.DNO = D.DNUMBER **AND** D.MGRSSN = M.SSN

  ACTION_PROCEDURE

  INFORM_MANAGER (D.MGRSSN)

3 ER Diagram

Entity: A "thing" in the real world (physical or conceptual).

Attribute: Particular properties of an entity.

Type of attributes:

simple vs composite

single-valued vs multivalued

stored vs derived

Entity type defines a set of entities that have the same attributes.

An entity type describes the schema for a set of entities that share the same structure.

An important constraint on the entities of an entity type is the **key** or **uniqueness constraint**.

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values).

Degree: the number of participating entity types.

Participation constraints: whether the existence of an entity depends on its being related to another entity via the relationship type.

Total (existence dependency)

Partial

Weak entity types: some entity types that may not have any key attributes of their own. Therefore, a weak eneity type always has a total participation constraint and a partial key.

ER-relational Mapping Algorithm (7 steps):

1. **Mapping of regular entity types**.

   For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E.

   Include only the simple component attributes of a composite attribute.

   Choose one of the key attributes of E as primary key for R.

   If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

   Note: foreign keys are not included yet.

2. **Mapping of weak entity types**.

   For each weak entity type W in the ER schema with owner entity type E, create a relation R, and include all simple attributes of W as attributes of R.

   Include as a foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).

   The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W.

3. **1 : 1 relationship types**.

   For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.

   Include the primary key of T as a foreign key in S. It is better to choose as entity type with total participation in R.

Include all the simple attributes of the 1:1 relationship type R as attributes of S.

Note: When both participations are total, we may merge the two entity types and the relationship into a single relation.

4. **1 : N relationship types**

   For each regular 1:N relationship type R, identify the relation S that represents the participating entity type at the *N-side* of R.

   Include the primary key of the relation T as a foreign key in S that represents the other entity type participating in R.

   Include any simple attributes of the 1:N relationship type as attributes of S.

5. **M : N relationship types**.

   For each M:N relationship type R, create a new relation S to represent R.

   Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.

   Include any simple attributes of the M:N relationship type as attributes of S.

   Note: We cannot represent an M:N relationship type by a single foreign key attribute.

   Note: The propagate (**CASCADE**) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to R, since each relationship instance has an existence dependency on each of the entities it relates.

6. **Multivalued attributes**.

   For each multivalued attribute A, create a new relation R.

   Include an attribute corresponding to A, plus the primary key attribute K as a foreign key in R.

   The primary key of R is the combination of A and K.

   Note: The propagate (**CASCADE**) option for the referential triggered action should be specified on the foreign key in the relation corresponding to the multivalued attribute.

7. **N-ary relationship types**.

   For each n-ary relationship type R, when $n > 2$, create a new relation S to represent R.

   Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.

   Include any simple attributes of the n-ary relationship type as attributes of S.

   The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.

4 Normal Form and Normalization

Informal measure of quality for relation schema design:

Semantics of the attributes.

Reducing the redundant values in tuples.

Reducing the NULL values in tuples.

Disallowing spurious tuples.

Four Guidelines:

1. Do not combine attributes from multiple entity types and relationship types into a single relation.
2. No update anomalies (insertion, deletion, and modification) occur in the relation.
3. Avoid placing attributes with NULLs in a base relation. If NULLs are unavoidable, make sure that they apply in exceptional cases only.
   NULLs can have multiple interpretations:
   Not applying to the tuples, unknown values, or known but absent values.
4. Relation schemas can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guqrantees that no spurious tuples (those represent wrong information that is not valid) are generated.

Functional dependencies:

If $X \to Y$, for any two tuples t_1 and t_2 such that $t_1[X] = t_2[X]$, we must have $t_1[Y] = t_2[Y]$.

$X \to Y$ in R doesn't imply that $Y \to X$ in R.

A functional dependency cannot be inferred automatically from a given relation but must be defined explicitly by someone who knows the semantics of the attributes of R.

Normalization of date is a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties.

One objective of the normalization is to ensure that the update anomalies do not occur.

Two properties must be preserved in the process of normalization:

Lossless join.

Dependency preservation.

Prime attribute: a member of any key.

Nonprime attribute: not a prime attribute.

First Normal Form (1NF):

Disallow multivalued and composite attributes.

Disallow relations within relations.

That is, the domains of attributes must include only atomic values.

Second Normal Form (2ND):

$X \to Y$ is a **full functional dependency**

if removal of any attribute A from X means that the dependency does not hold any more.

R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

Third Normal Form (3NF):

$X \to Y$ is a **transitive dependency**

if there is a set of attributes Z that is not a subset of any key of R, and both $X \to Z$ and $Z \to Y$ hold.

R is in 3NF if it is in 2NF and no nonprime attribute of R is transitively dependent on the primary key.

General definitions of 2NF and 3NF:

Take all candidate keys into account.

R is in 2NF if every nonprime attribute A in R is not partially dependent on any key of R.

R is in 3NF if, whenever $X \to Y$ holds in R, either X is a superkey of R *or* Y is a prime attribute of R.

Note: An attribute that is part of any candidate key will be considered as a prime.

Stay hungry, stay foolish.

Good Luck!