CS442

NOTE: Diagrams in notebook

11/29/2021

Schema Refinement Tutorial (Part II)

3NF Decomposition Exercise

- R(ABCDE) has FD F = {ABCD -> E, E -> D, A -> B, AC -> D}
- Does R satisfy 3NF? If not, decompose until it does.
- Step 1: Find candidate keys

L	M	R
AC	BDE	

  - AC+ = ABCDE, AC is a superkey, and since both AC need to be included, it is minimal
- Step 2: 3NF Check: For each FD X -> Y, either X is a CK, or Y is in a CK
  - ABCD -> E: Neither case holds, so violates 3NF
  - Decompose:
  - Step 1: Find minimal cover F' of F
    - Step 1.1: Minimize RHS of each FD (if X -> YZ, then replace X -> YZ with X -> Y and X -> Z)
      - Already done, F' = F
    - Step 1.2: Minimize LHS of each FD (if X -> Y and XY -> Z, then replace XY -> Z with X -> Z)
      - Replace ABCD -> E with ACD -> E since A -> B
      - Replace ACD -> E with AC -> E since AC -> D
      - F' = {AC -> E, E -> D, A -> B, AC -> D}
    - Step 1.3: Remove redundant FDs (if X -> Y, Y-> Z, and X -> Z, remove X -> Z)
      - Remove AC -> D because AC -> E and E -> D.
      - F' = {AC -> E, E -> D, A -> B}
  - Step 2: BCNF Decomposition R(ACBDE) in BCNF according to F' = {AC -> E, E -> D, A -> B}
    - Step 2.1 Check if R satisfies BCNF
      - AC+ = ABCDE, AC is a superkey, satisfies BCNF
      - E+ = ED, E is not a superkey, violates BCNF

- A+ = AB, A is not a superkey, violates BCNF
- R violates BCNF because of E -> D and A -> B

■ Step 2.2 BCNF Decomposition

- Split on E -> D
- R1 = R - Y: ABCE {A -> B, AC -> E}
  - AC+ = ACE, AC is not a superkey, violates BCNF
    - Split on A -> B
    - R1 = R - Y: ACE {AC -> E}
      - AC+ = ACE, AC is a superkey, satisfies BCNF
    - R2 = XY: AB {A -> B}
      - A+ = AB, A is a superkey, satisfies BCNF
- R2 = XY: ED {E -> D}
  - E+ = ED, E is a superkey, satisfies BCNF

■ Decomposition: [(AB), (ACE), (ED)]

○ Step 3: Dependency-preserving decomposition (If X -> Y is not preserved, add (XY) to list) according to F'

■ F' = {AC -> E, E -> D, A -> B}

■ Everything is preserved!

11/22/2021

<u>Schema Refinement Tutorial (Part I)</u>

Exercise

- Given FD = {AB -> CD, C -> DE, C -> F, E -> G, A -> B. Prove that {A -> FG} is logically implied by FD
- A+ = ABCDEFG from A -> B, AB -> CD, B -> DE, C -> F, E -> G
- Since FG $\subseteq$ A+, A -> FG

Using FDs to Find Keys

- A superkey K of a schema R is a set of attributes in R st K -> R holds. A candidate key is minimal.

Exercise

- Consider the relation schema R(ABCD) with F = {A -> C, B -> D}. Is {A,B} a candidate key?
- AB+ = ABCD. AB is a superkey! Is AB minimal?
- A+ = AC, B+ = BD. Neither of these are superkeys. AB is minimal! AB is a candidate key.

Exercise

- R(ABCD), F = {D -> B, AB -> D, AB -> C, C -> A}
- Candidate keys? AB+ = ABCD, BC+ = ABCD, AD+ = ABCD, CD+ = ABCD
- Checking of 3NF
    - Satisfaction Check: For each FD X -> Y, X is a cand key or Y is in atleast one cand key
    - D -> B holds since B is in two cand keys, AB -> D holds since AB is a cand key,
    - AB -> C holds since AB is a cand key, C -> A holds since B is in two cand keys.
    - Therefore, R satisfies 3NF
- Checking of BCNF
    - Satisfaction Check: For each X -> Y, X is a superkey
    - D -> B: D+ = DB. D is not a superkey; this violates BCNF
    - AB -> D, AB is a superkey; this satisfies BCNF
    - AB -> C, AB is a superkey; this satisfies BCNF
    - C -> A: C+ = AC. C is not a superkey; this violates BCNF
    - Split on D -> B
        - LHS: R - Y, RHS: XY
        - LHS: ACD {C -> A}
            - Satisfaction Check: C+ = CA. C is not a superkey, violating BCNF
            - Split on C -> A
                - LHS: CD {}
                    - Satisfaction Check: Good by default
                - RHS: AC {C -> A}
                    - Satisfaction Check: C+ = CA. C -> A holds since C is a superkey
        - RHS: BD {D -> B}
            - Satisfaction Check: D+ = DB. D -> B holds since D is a superkey
    - BCNF Decomposition: AC, BD, CD

11/19/2021

<u>3NF Decomposition (Part I)</u>

Good Decomposition

- Lossless (can be obtained by BCNF decomposition)
- Dependency Preserving (all FDs still hold)

Dependency Preserving Decompostion

- Projection of set of FDs F
    - If R is decomposed into X and Y, then the projection of F on X (denoted F_X) is the set of FDs U -> V in F+ st all of the attributes U,V are in X (*similarly for F_Y*).
        - Ex: F = {A -> B, A -> C, D -> E}, X = {ABC}, Y = {CDE}
        - F_X = {A -> B, A -> C}
        - F_Y = {D -> E}
- Decomposition of R into X and Y
    - This is dependency preserving if $(F_X \cup F_Y)^+ = F^+$
    - Minimal Cover for a Set of FDs
        - G is the **minimal cover** of a set of FDs F if G satisfies the following three conditions:
            - RHS of each FD in G is a single attribute
            - F+ = G+
            - G is minimal (if we modify G by deleting an FD in G, G+ changes).
        - Intuitively, every FD in G is needed, and is as small as possible in order to get the same closure as F
    - Finding Minimal Cover
        - Step 1: Minimize RHS of FDs so they only contain single attributes
            - Change X -> YZ into X -> Y and X -> Z
        - Step 2: Minimize LHS
            - If there are two FDs: A -> B and ABX -> Z, replace ABX -> Z with AX -> Z
        - Step 3: Remove Redundant FDs
            - If X -> Y can be inferred from other FDs, remove it
    - Example
        - R(ABCDE), F = {A -> D, BC -> AD, C -> B, E -> A, E -> D}
        - What is the minimal cover of F?
            - Step 1: Minimize RHS
                - F = {A -> D, BC -> A, BC -> D, C -> B, E -> A, E -> D}

- Step 2: Minimize LHS
  - F = {A -> D, C -> A, C -> D, C -> B, E -> A, E -> D}
  - F = {A -> D, C -> A, C -> B, C -> D, E -> A, E -> D}
- Step 3: Remove Redundant FDs by Transitivity
  - C -> A, A -> D: C -> D is redundant
  - E -> A, A -> D: E -> D is redundant
- Minimal Cover: **F' = {A -> D, C -> A, C -> B, E -> A}**

○ 3NF Decomposition
- Step 1: Find the Minimal Cover F' of F.
- Step 2: Generate a BCNF decomposition {R1, ..., Rn} of R wrt F'
- Step 3: Identify the dependencies N in F' that is not preserved by BCNF decomposition {R1, ..., Rn}
  - ie. Those dependencies in F' whose attributes are not included in any table in {R1, ..., Rn}
- Step 4: For each X -> Y in N, create a relation schema XY and add it to {R1...Rn}
- Notes:
  - Step 2 guarantees it is a lossless decomposition
  - Step 3 and 4 ensure it is a dependency-preserving decomposition
    - Both steps ensure $F_{R1} \cup ... \cup F_{Rn} = F'$
    - Thus it must be true $(F_{R1} \cup ... \cup F_{Rn})^+ = F'^+$
  - 3NF decomp. is guaranteed to be lossless and dependency preserving

○ 3NF Example
- R(ABCDE), F = {A -> D, BC -> AD, C -> B, E -> A, E -> D}
- Decompose R in a lossless-join, dependency-preserving way (ie. decompose R into 3NF tables).
  - Step 1: Find minimal cover F' of F (similar ex done in prev. example)
    - F' = {A -> D, C -> A, C -> B, E -> B, E -> D}
  - Step 2: Generate BCNF decomposition wrt F'
    - Calculate candidate keys:

L	M	R

CE	A	BD

- CE+ = CEABD: CE is the candidate key

- Step 2.2: BCNF Decomposition
  - ABCDE {A -> D, C -> A, C -> B, E -> B, E -> D}
    - None of the closures (A+,C+,E+) satisfy BCNF
    - Split on A -> D: L = R1, R = R2
      - R1 = ABCE {C -> A, C -> B, E -> B}
        - None of the closures (C+,E+) satisfy BCNF
        - Split on C -> A: L = R11, R = R12
          - R11 = BCE {C -> B, E -> B}
          - None of the closures (C+,E+) satisfy BCNF
          - Split on C -> B: L = R111, R = 112
          - R111 = CE {} holds by default
          - R112 = BC {C -> B} holds,C+ = CB
          - R12 = AC {C -> A} holds, C+ = CA
      - R2 = AD {A -> D} holds, A+ = AD
  - BCNF Decomposition: **AD, AC, BC, CE**
- Step 3: Identify all FDs N ⊆ F' which do not hold by BCNF decomp.
  - A -> D: holds in AD
  - C -> A: holds in AC
  - C -> B: holds in BC
  - E -> B, E -> D: do not hold in any
- Step 4: Add tables for FDs in N into BCNF decomposition
  - Final 3NF Decomposition: {AD, AC, BC, CE, EB, ED}
    - First 4 from BCNF decomp
    - Last 2 for dependency-preservation

- BCNF vs 3NF Decomposition
  - For any given R, there will always be a 3NF decomposition
    - But it may not have a BCNF decomposition
    - BCNF is stricter than 3NF

- A BCNF decomposition is always a 3NF decomposition

11/17/2021

BCNF Decomposition

Problems with Decomposition

- May be impossible to reconstruct the original relation (lossiness)
- Dependency checking may require joins

Good Decomposition

- Lossless, and dependency preserving

	BCNF Decomposition	3NF Decomposition
Data Redundancy	NONE	May still have some
Lossless	Guaranteed	Guaranteed
Dependency-Preserving	Not guaranteed	Guaranteed

Lossy Decomposition

- Join result of the tables after decomposition is not the same as the original dataset

Lossless Decomposition

- Join result of the tables after decomposition is the same as the original dataset

How can we decompose R so that it is lossless?

- Given a table R and a set of FDs F of R, the decomposition of R into X and Y is lossless with respect to F iff the F+ satisfies: X ∩ Y -> X or X ∩ Y -> Y. The common attributes of X and Y is a superkey of either X or Y
- If W -> Z holds over R and W ∩ Z is empty, then
    - Decompose R into two tables R1 = R - Z and R2 = WZ (concatenate the two)
    - The decomposition (R-Z,WZ) are guaranteed to be lossless (as R-Z and WZ joins at W, and W -> Z)

Exercise

- R = ABCDE, FDs F = {AB -> C, C -> E, B -> D, E -> A}
- R is decomposed into R1 = BCD and R2 = ACE

- Is (R1,R2) a lossless decomposition?
    - Step 1: Find common attributes of R1 and R2
        - R1 ∩ R2 = {C}
    - Step 2: Find whether C -> BCD or C -> ACE (C is a superkey of either R1 or R2)
        - C+ = CEA: C -> ACE; C is a superkey of R2
    - Thus (R1,R2) decomposition is lossless

Algorithm of BCNF Decomposition

- Step 1: Ensure all FDs in F contain only a single attribute at the right-hand side
    - This is always doable by decomposition rule
    - For example, if you have AB -> CD, split it into AB -> C and AB -> D
- Step 2:
    - Identify all FDs F' ⊆ F that hold on R
        - An FD f holds on R if R contains all attributes in f
        - This substep is not needed for the first round where R is the original dataset, but is required for all the decomposed tables
    - Check if R satisfies BCNF according to F'
    - If not, for any X -> Y in F' that violates BCNF (i.e. X is not a superkey of R), decompose R into R1 = R - Y and R2 = XY
- Repeat Step 2 on R1 and R2 until all the decomposed tables satisfy BCNF
    - BCNF decomposition is guaranteed to be lossless!

Exercise

- R=(CSJDPQV), Primary key: C, FDs: {JP -> C, SD -> P, J -> S}
- Does R satisfy BCNF?
    - JP -> C: JP+ = JPCSDQV; JP is a superkey. FD satisfies BCNF.
    - SD -> P: SD+ = SDP; SD is **not** a superkey. FD **violates** BCNF.
    - J -> S: J+ = JS; J is **not** a superkey. FD **violates** BCNF.
- If not, decompose R into BCNF tables
    - R = CSJDPQV -- JP -> C good
      SD -> P bad, J -> S bad no J for JP -> C, J -> S, SD is a superkey
      Split with SD -> P: R1 = CSJDQV R2 = SDP -- SD -> P holds, SD+ = SDP, sats. BCNF
      no P for JP -> C, SD -> P

J -> S: J+ = JS; violates BCNF

Split new R1 with J -> S: R11 = CJDQV R12 = JS -- J -> S holds, sats. BCNF

no valid FD, so sats. BCNF

- Final Decomposition: (CJDQV, JS, SDP)

11/15/2021

Normal Forms (NF)

Types: 1st, 2nd, 3rd, Boyce-Codd (BCNF or 3.5 NF)

- The higher the normal form is, the stricter the constraints are that are put on the database
  - Order of checking normal forms: 2nd NF, 3rd NF, BCNF.
- First Normal Form
  - The definition of the relational schema
  - A relational schema is in 1NF iff the **domains of all attributes of R are atomic**
  - A domain is atomic if elements of the domain are considered to be indivisible units.
- Second Normal Form
  - Partial Dependency: An FD X->Y is said to be a partial dependency if there exists an FD Z->Y such that $Z \subseteq X$ (i.e., Z is a subset of X).
  - A relation schema R is in 2NF if there are **no partial dependencies**
  - Check violation of 2NF:
    - Given relation R and its FDs F, check whether some attributes Y partially depends on a candidate key X
      - In other words, check if there exists an FD X->Y such that:
        (1) X is a subset of candidate keys of R;
        (2) Y is a non-key attribute (i.e., Y does not appear in any candidate key)
    - If there is such FD, then R violates 2NF (because X->Y is a partial dependency)
- Third Normal Form
  - Transitive Dependency: There exists transitive dependency when a non-key attribute A determines another non-key attribute B.
    - K-> A->B (K: the key)
  - Relation R is in 3NF if: **No partial dependency** AND **no transitive dependency**

        - i.e., the relation R is in 2NF, and for each candidate key K of R, all non-key attributes of R are directly dependent on K.
    - Check violation of 3NF:
        - There are two cases when an FD: X-> Y violates 3NF:
            - Case 1: X ⊆ candidate key K (this is a partial dependency, as K -> Y too)
            - Case 2: X is not a subset of a candidate key and Y is a non-key attribute (i.e., Y is not included in any candidate key)
                - This is a transitive dependency, as it has dependencies K -> X -> Y.
- Shortcut Rules
    - Shortcut Rule 1: If all candidate keys are singleton keys (i.e., only contain one attribute), then R must satisfy 2NF
    - Shortcut Rule 2: If all attributes are part of some candidate keys, then R must be 2NF & 3NF (WHY?)
- Boyce-Codd Normal Form (BCNF/3.5NF)
    - Relation R with FDs F is in BCNF if, for all X -> Y in F, it satisfies the following condition:
        - X is a superkey for R.
- Examples of 3NF vs BCNF
    - 1. R(ABC), R has a key = (A, B), F = {C->B}
    - 1. Does R satisfy BCNF?
        - BCNF Checking: Whether for any FD X -> Y, X is a superkey
        - C -> B: Is C a superkey? C+ = CB; C is **not** a superkey
        - Therefore R violates BCNF
    - 1. Does R satisfy 3NF?
        - 3NF Checking: If it's true that for any FD X -> Y, either X is a candidate key, or Y is included in a candidate key
        - FD: C -> B: Is C a candidate key? No, C is not even a superkey as shown above
        - C -> B: Is B included in a candidate key? Yes, B is included in (A,B)
        - Therefore R is in 3NF
    - 2. R(ABC), F = {AB -> C, C -> A}, What normal form does R satisfy?
        - Step 1: Find candidate keys

L	M	R

B	A, C	

- Candidate keys must include ‘B’ attribute
- B+ = B; B is not a candidate key
- AB+ = ABC; AB is a candidate key
- BC+ = ABC; BC is a candidate key as well
- Candidate Keys: **AB, BC**
- Non-key Attributes: []

■ Step 2: Check 2NF
- Check violation: Any FD X -> Y s.t. X is a subset of CK, and Y is a non-key attribute
- AB -> C: AB is a CK, so it will not violate 2NF
  C -> A: C is a subset of a CK, but A is included in a CK (not a non-key attr), so it will not violate 2NF either
- **R satisfies 2NF**.
- or use shortcut rule #2 (if all attributes are part of CKs, then R must satisfy 2NF and 3NF)

■ Step 3: Check 3NF
- Check: For all FD X -> Y, either X is a CK or Y is included in a CK
- AB -> C: AB is a CK, so it satisfies 3NF
  C -> A: C is not a CK, but A is included in a CK, so it also satisfies 3NF
- **R satisfies 3NF**.

■ Step 4: Check BCNF
- Check: For all FD X -> Y, X must be a superkey
- AB -> C: AB is a CK, so it satisfies BCNF
  C -> A: C+ = {CA}, so C is not a superkey
- **R does not satisfy BCNF**.

■ Conclusion: R satisfies 3NF

11/12/2021

Functional Dependencies (II)

Exercises:

- Given: Relation R = {A,B,C,D,E,F,G,H,I} and FDs A → B, A → C, CG → H, CG → I, B → H
  - Prove A → H: Transitivity -> A → B, B → H, therefore A → H
  - Prove AG → I: Augmentation -> A → C therefore AG → CG,
    Trans -> CG → I, therefore AG → I
  - Prove CG → HI: Union -> CG → H, CG → I, therefore CG → HI
- Given schema Contracts(cid, sid, jid, did, pid, qty) has FDs:
  - C is the key: C → CSJDPQ, JP → C, SD → P
  - Prove SDJ is a superkey for Contracts:
    - SD → P therefore SDJ → JP, therefore SDJ → C
    - SDJ → C, therefore SDJ → CSJDPQ. Therefore SDJ is a superkey.

Computer FD/Attribute Closure

- F+: All FDs that are implied from some given FDs F (including trivial dependencies like A -> A)
- Ex: R(ABCDE), F = {A → D, D → B, B → C, E → B}
  - A+ = ADBC from A → D, D → B, B → C
  - D+ = DBC from D → B, B → C
  - E+ = EBC from E → B, B → C
  - ACE+ = ACEDB from A → D, D → B -> ACE is a superkey of R
- Ex: R(ABCDE), F = {B → CD, D → E, B → A, E → C, AD → B}
  - Can B → E be inferred by F? ie Is B → E in F^+?
  - Easier answer: Is E in B^+? B^+ = BCDEA. Since E in B^+, B → E!
- How can we check if the superkey X is a candidate key? Check if it's minimal.
  - We must check if the closures of any of its subsets are also superkeys.
- R = {ABCDE}, F {B → CD, D → E, B → A, E → C, AD → B }
  - Is D a superkey of R? D+ = DEC. D+ is not ABCDE, so its not a superkey
  - Is B a superkey? B+ = BCDEA. B+ is ABCDE, so its a superkey
    - Is B a candidate key? **YES**, it only has one attribute
  - Is AD a superkey? AD+ = ADBEC. Since B is a superkey, AD is a superkey too

        - Is AD a candidate key? Check subsets: A+ = A, D+ = DEC.
            - Neither A nor D is a superkey, so **YES**, AD is a candidate key

How to Determine Candidate Keys

- Group attributes into three categories:
    - L: Attributes that only appear on the left of the arrow (A from A → B)
    - R: Same but only on the right of the arrow (B from A → B)
    - M: Attributes that appear on both sides of the arrow (M from A → M, M → B)
        - Attributes in L will always be a part of candidate keys
        - Attributes in R will never be a part
- R(ABCD), FDs(AB → C, C → B, C → D)
    - L: A M: B,C R: D
    - A+: A, so A itself **is not** a candidate. Move on to LM combo
    - AB+: ABCD, so AB **is** a candidate key
    - AC+: ACBD, so AC **is** a candidate key
    - ABC+: don't consider since this is a superset of AB
    - Therefore: AB and AC are candidate keys

11/10/2021

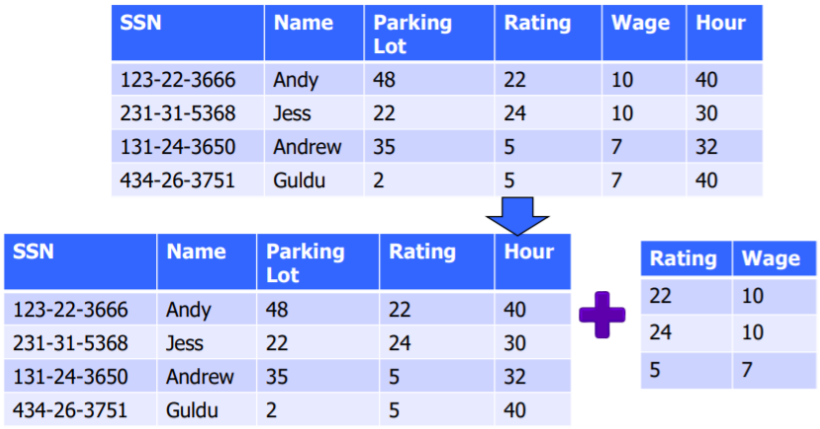
<u>Functional Dependencies</u>

Data Redundancy

- MailingAddress(<u>Name</u>, Dept, Address)
    - Dept is always associated with Address -- redundant
- Person(<u>SSN</u>,Name, Address, <u>Hobby</u>)
    - Assume one person can only have one address
    - Name is always associated with Address -- redundant
- Problems That Arise Because of Redundancy
    - UPDATE ANOMALIES
        - What if Allen moves to a new place, but only one of his records is updated?
    - INSERTION ANOMALIES
        - What if a new hobby for Allen is inserted, but his address has changed?
    - DELETION ANOMALIES

        - What if Bob is no longer interested in Video Games; should we also delete his address and other info from the database?

Functional Dependency (FD)

- FD: A set of attributes can determine another set of attributes
    - So if SSN determines Address, we say that there's a functional dependency: Address depends on SSN.
- X -> Y: X determines Y
    - Holds over relation schema R if the following holds
        - For any two records t_0,t_1 in R, if $t_0[X] = t_1[X]$, then $t_0[Y] = t_1[Y]$.
        - i.e., all records that have the same X values always have the same Y values (but not necessarily vice versa)
    - From above, FD: SSN -> Address (any two records of the same SSN must have the same address)
        - However, **X -> Y does not imply Y -> X**, Address -/> SSN (there may be multiple people living at the same address
    - What are the possible FDs of Hourly_Emps(<u>ssn</u>, name, lot, rating, wage, hours)?
        - *ssn* is the key: S -> SNLRWH
        - *rating* determines wage: R -> W
        - *lot* determines lot: L -> L (Trivial Dependency)
- Eliminating Redundancy by Decomposition
    - 

SSN	Name	Parking Lot	Rating	Wage	Hour
123-22-3666	Andy	48	22	10	40
231-31-5368	Jess	22	24	10	30
131-24-3650	Andrew	35	5	7	32
434-26-3751	Guldu	2	5	7	40

SSN	Name	Parking Lot	Rating	Hour
123-22-3666	Andy	48	22	40
231-31-5368	Jess	22	24	30
131-24-3650	Andrew	35	5	32
434-26-3751	Guldu	2	5	40

Rating	Wage
22	10
24	10
5	7

    - Rating, Wage goes to Rating, (Key, Value) for Rating, Wage
    - <u>Update Anomaly</u>: What if we want to modify the Wage of the 3rd tuple?

        - Wage is updated in the RW table (including the wage of the 4th tuple).
    - Insertion Anomaly: What if we want to insert an employee but don't know the hourly wage for his/her rating?
        - If the rating exists in the RW table, we only need to insert the employee's rating in the SNPRH table.
    - Deletion Anomaly: What if we delete all employees with a rating of 5?
        - The wage information of rating 5 still exists in the RW table.
- Use FDs to Determine Keys
    - If "K -> All Attributes of R", then K is a superkey for R
        - K is a superkey because it does not require K to be *minimal*.
        - FDs are a generalization of keys
            - FDs are not necessarily key constraints
- Use FDs to Infer Additional FDs
    - Given some FDs, we can infer additional FDs:
        - Distributive Property:
            - title -> (studio, star) implies title -> studio and title -> star
            - title -> studio and title -> star implies title -> (studio, star)
        - Transitive Property:
            - title -> studio, studio -> star implies title -> star
    - An FD F_{new} is implied by a set of FDs F **if** F_{new} holds whenever all FDs in F hold.
    - F^+ (closure of F): The set of all FDs that are implied by F (includes trivial dependencies).
- Rules of Inference: Armstrong's Axioms (AA)
    - Given X, Y, Z, are sets of attributes
    - Reflexivity: If $X \subseteq Y$, then Y -> X
        - Ex: X = {AB}, Y = {ABCD}, ABCD -> AB
    - Augmentation: If X -> Y, then XZ -> YZ for any Z
        - Ex: X = {A}, Y = {B}, Z = {CD}. If A -> B, then ACD -> BCD
    - Transitivity: If X -> Y and Y -> Z, then X -> Z
        - Ex: X = {A}, Y = {B}, Z = {C}. If A -> B and B -> C, then A -> C
    - Union/Decomposition: If X -> Y and X -> Z, then X -> YZ
      If X -> YZ, then X -> Y and X -> Z

11/01/2021

JDBC Programming

Pure SQL: Queries typed at an SQL prompt.

- SQL is a non-procedural language
- Sophisticated applications are often implemented by using SQL + a programming language

Embedded SQL: SQL can be embedded within procedural programming languages

- Customized applications
- Background applications running without user intervention
- Combining database tools with programming tools
- Two types of embedding:
    - Type 1: Low-level embedding (C/C++)
    - Type 2: ODBC - Open Database Connectivity
        - SQL queries are sent from the program to the database as strings
        - Results are returned as an array or a list

10/25/2021

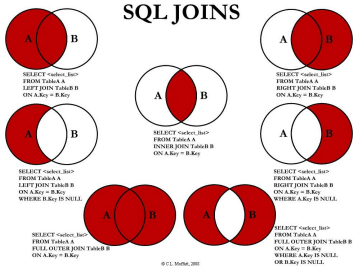
SQL: Null Values, Join, and Sorting

NULL Values

- Special values that SQL provides for the values that are unknown or inapplicable
- Example:
    - Rating for a new sailor (which is not available)
        - INSERT INTO Sailors (sid, snake, rating, age) VALUE (101, 'John', 18);
        - Does not have a rating yet ^^
    - Maiden name for male students
- Introduce the third-value logic (true, false, unknown), to handle NULL values
- Comparison with NULL values
    - The result of comparison between two NULL values or between a NULL and known will always result as UNKNOWN
        - Ex: rating>8 returns UNKNOWN if a rating is NULL

        - The result of a comparison involving NULL is not a boolean value - it is a non-value
    - In SQL: IS NULL, or IS NOT NULL. **Never use = NULL or <> NULL.**
- Logical Connectives with UNKNOWN
    - NOT unknown -> unknown
    - <expression> OR unknown -> true if <expression> is true, unknown if <expression> is false
    - <expression> AND unknown -> unknown if <expression> is true, false if <expression> is false
- Impact on SQL Constructs
    - NULL values are present in SELECT DISTINCT result
        - All null values are considered as identical and duplicate, so only one will show
    - Arithmetic operations (+, -, *, /) return NULL when executed on NULL values
- In Aggregate Queries
    - COUNT, SUM, AVG, MIN, MAX on **specific columns** do not consider NULL values
    - COUNT(*) is the exception and does **include** NULL values
    - Special case: if all values on the attribute are NULL, then these operators return NULL
        - Exception: COUNT(A) = 0 if all values of attribute A are NULL
- Disallow NULL Values
    - Null values can be disallowed by specifying NOT NULL as part of the definition in CREATE SQL statement
        - Example:
            - CREATE TABLE User(ID INT PRIMARY KEY, Name CHAR(10) NOT NULL);
        - Implications:
            - SQL enforces NOT NULL automatically on the primary key
            - Candidate keys allow exactly one (1) NULL value (more than one would result in duplicates

Inner/Outer Join

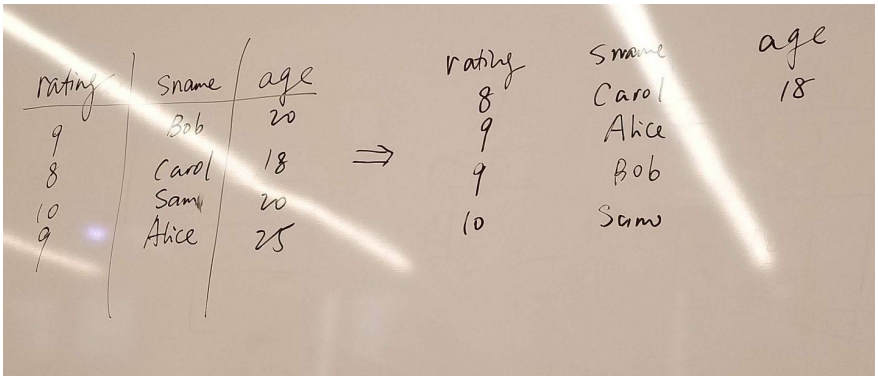


- Inner Join
    - Syntax: SELECT column_name(s)
        - FROM table1 INNER JOIN table2
        - ON table1.column_name = table2.column_name;
    - Only the rows that match the search conditions are returned
    - Same as equal join

- Left Whole Join (Left Outer Join)
    - LEFT JOIN instead of INNER JOIN
    - Returns all records from the left table (A) and the matching records from the right table (B).
    - The result is 0 records from the right side if there are no matches
- Right Whole Join (Right Outer Join)
    - RIGHT JOIN instead of INNER JOIN
    - Returns all records from the right table (B) and the matching records from the left table (A).
    - The result is 0 records from the left, and all of the right side if there are no matches
- Full Join
    - Syntax: tableA FULL OUTER JOIN table
    - A full outer join is the combination of a left join and a right join.
    - The full outer join includes all rows from the joined tables whether or not the other table has the matching row.

Sorting

- Syntax: SELECT col1, col2, ...
    - FROM table_name
    - ORDER BY col1, col2, ... ASC|DESC;
        - By default: Ascending order
        - Ordering attributes must appear in the SELECT clause
- Example:
    - SELECT S.rating, S.sname, S.age
    - FROM Sailors NATURAL JOIN Reserves R NATURAL JOIN Boats B
    - WHERE B.color = ‘red’
    - ORDER BY S.rating, S.sname #first orders by rating, and then second by name

- 

  -

10/22/2021

In-Class Exercises

10/20/2021

SQL: Aggregate Queries (Part V)

- Aggregate Function
  - Input: A collection of values
  - Output: A single value (aggregated result)
  - Types of aggregate functions:
    - SUM, AVG, MIN, MAX, COUNT
    - summation, average value, min value, max value, number of rows (including NULL rows)
  - Used ONLY in the SELECT and HAVING clauses (HAVING clause will
  - be covered later today).
- Examples:
  - COUNT(*)
  - Can use DISTINCT, ALL for COUNT, SUM, and AVG (**default** = **ALL**)
    - COUNT(DISTINCT A) or COUNT(A), same for SUM and AVG
  - MIN(A) and MAX(A)
- Properties for Input Data
  - SUM and AVG
    - Only accepts numerical values as input

    - MIN, MAX, COUNT
        - Accepts both numerical and categorical values as input
    - MIN, MAX, AVG, SUM
        - Only considers non-null values
    - COUNT
        - Includes NULLs
- Examples:
    - AVG (ALL A) = (10k+10k+20k+30k)/4 = 17.5k
        - AVG (DISTINCT A) = (10k + 20k +30k)/3 = 20k
    - Find the name and age of the oldest sailor(s).
        - SELECT S.sname, S.age
        - FROM Sailors S
        - WHERE S.age = (SELECT MAX (S2.age)
            - FROM Sailors S2);
- GROUP BY
    - GROUP BY: groups the rows of the same values into groups
        - SELECT att1, att2, ..., attk, aggregate_function
        - FROM tables
        - [WHERE qualifications]
        - GROUP BY att1, att2, ... attn;
        - k <= n
    - “For each” → GROUP BY
    - For each rating level, find the age of the youngest sailor -- Sailors(<u>sid</u>, sname, age, rating).
        - SELECT MIN(age) as Min_age
        - FROM Sailors
        - GROUP BY rating;
    - For each rating, find the average age of the sailors
        - SELECT AVG(age)
        - FROM Sailors
        - GROUP BY rating;
    - For each rating, find the age of the youngest sailor among all sailors older than 18

        - SELECT MIN(age) as Min_age
        - FROM Sailors
        - WHERE age > 18
        - GROUP BY rating;
    - GROUP BY with Join
        - Boats(<u>bid</u>, bname, color), Sailors(<u>sid</u>, snake, rating, age), Reserves(<u>sid, bid</u>, day)
        - SELECT B.bid, COUNT(*)
        - FROM Boats B NATURAL JOIN Reserves R
        - WHERE color = ‘red’
        - GROUP BY R.bid;
- HAVING
    - HAVING Clause: Conditional clause with GROUP BY Clause
        - Filters groups based on GROUP BY results
        - Requires a GROUP BY clause to be present
        - Attributes in group-qualification of the HAVING clause must either be an aggregate operation or appear in att1, att2, ... attn of the GROUP BY clause
    - Syntax
        - SELECT att1, att2, ..., attk, aggregate_function
        - FROM tables
        - [WHERE qualifications]
        - GROUP BY att1, att2, ... attn
        - HAVING group-qualifications;
    - For each rating group that has more than 10 sailors, find its lowest rating.
        - SELECT MIN(rating) as min_rating
        - FROM Sailors
        - GROUP BY rating
        - HAVING COUNT(*) > 10;
    - HAVING vs WHERE
        - WHERE filters individual records, HAVING filters groups of records
        - **Rule**

- If the selection condition is specified on the aggregate values (eg. min/max), use HAVING
- Otherwise, the selection condition is specified on the individual records, and so use WHERE

- Use GROUP BY and HAVING Clauses for the Division operator
  - Find the name of sailors who've reserved all boats.
    - SELECT S.sname
    - FROM Sailors S NATURAL JOIN Reserves R
    - GROUP BY S.sname, S.sid
    - HAVING COUNT(DISTINCT R.bid) = (SELECT COUNT (*) FROM Boats);
  - Can we use *sid only* in GROUP BY clause?
    - **NO**,
  - Can we use *sname only* in GROUP BY clause?
    - **NO**, sailors can have same name
  - Can we remove DISTINCT from HAVING clause, assuming the same sailor can reserve the same boat at different days?
    - **NO**, same sailor can remove same boat '# of boats' times

10/18/2021

Announcement: Assignment 4 due Nov 1

MIDTERM: Oct 27: ER diagram design, transforming ER diagram to relational tables,

Oct 29: Queries (relational algebra, SQL)

CHEAT SHEET ALLOWED - one-sided, standard size

<u>SQL: Nested Queries (Part IV)</u>

- One of the most powerful features in SQL
- <u>Subquery</u> (Inner Query; Nested Query): A query within another SQL query and embedded within the WHERE clause.
  - SELECT …
  - FROM …
  - WHERE expression IN/EXISTS/<op> [ANY/ALL] (SELECT …
    - FROM …

        - WHERE ...);
- Subqueries must be enclosed within parentheses
- A subquery can only have one attribute in the SELECT clause, unless multiple attributes are in the main query for the subquery to compare its selected attributes
- How to Connect Subquery with Outer Query
    - In WHERE clause of main query:
        - in: WHERE <attributes> [NOT] IN (subquery)
            - IN: a set operation that checks set membership
            - rewrite INTERSECT using IN:
                - SELECT R.sid
                - FROM Boats B NATURAL JOIN Reserves R
                - WHERE B.color = 'red' AND R.sid IN (
                    - SELECT R2.sid
                    - FROM BOATS B2 NATURAL JOIN Reserves R2
                    - WHERE B2.color = 'green');
        - exists: WHERE [NOT] EXISTS (subquery)
            - EXISTS: Boolean operator that checks value existence
            - WHERE EXISTS returns *true* if the result of the subquery is not empty; if empty then false
            - rewrite INTERSECT using EXISTS:
                - SELECT R.sid
                - FROM Baots B NATURAL JOIN Reserves R
                - WHERE B.color = 'red' AND EXISTS (
                    - SELECT R2.sid
                    - FROM Boats B2 NATURAL JOIN Reserves R2
                    - WHERE R.sid=R2.sid and B2.color = 'green');
            - If there is no R.sid = R2.sid, then the output would be empty
        - op: WHERE expression op [ANY | ALL] (subquery)
            - ANY operator
                - Syntax: v op ANY S
                - Statement: v is <less than, greater than, equal to> ANY elem in S

        - returns true if at least one element t is in S such that v <op> t is true
        - Basically an OR
    - ALL operator
        - Syntax: v op ALL S
        - Statement: v is <less than, greater than, equal to> ALL elems in S
        - Basically an AND
- Find the names of sailors who've reserved all boats = Find the sailors for whom there is no such boat they have not reserved.
    - SELECT S.sname
    - FROM Sailors S
    - WHERE NOT EXISTS (SELECT B.bid
        - FROM Boats B
        - WHERE NOT EXISTS (SELECT R.bid
            - FROM Reserves R
            - WHERE R.bid = B.bid AND R.sid = S.sid));

10/13/2021

SQL: The Query Language (Part II)

Set Operations

- Union
    - Subquery1 UNION Subquery2
        - The two subqueries must be valid queries (SELECT-FROM block w optional WHERE)
    - Union must be union-compatible
        - The two subqueries must have the same attributes in the SELECT clause
    - UNION excludes duplicates
        - To include duplicates, use UNION ALL
    - Union of Two SELECT * Clauses
        - SELECT * FROM T1 UNION SELECT * FROM T2

        - The records of the same key but different non-key values are considered as different and thus are added into the union result
            - Key no longer can be used as key -- table cannot physically exist
            - To save it, must change key to new minimal key
    - Connective Priorities in SQL: NOT > AND > OR
    - No semicolon until the very end -> S1 UNION S2;
        - not S1; UNION S2;
    - Find ID of sailors who've reserved a red or green boat (**without set ops**)
        - SELECT R.sid FROM Boats B, Reserves R
        - WHERE R.bid=B.bid AND (B.color='red' OR B.color='red');
    - Same problem (**with set ops**)
        - SELECT R.sid FROM Boats B NATURAL JOIN Reserves R
        - WHERE B.color='red'
        - UNION
        - SELECT R.sid FROM Boats B NATURAL JOIN Reserves R
        - WHERE B.color='green';
- Intersect
    - Subquery1 INTERSECT Subquery2 (same as union)
        - The two subqueries must be valid queries (SELECT-FROM block w optional WHERE)
    - INTERSECT must be union-compatible (same as union)
        - The two subqueries must have the same attributes in the SELECT clause
    - INTERSECT excludes duplicates too
        - To include duplicates, use INTERSECT ALL
    - No semicolon until the very end
    - Find ID of sailors who've reserved a red and green boat (**without set ops**)
        - SELECT R.sid
        - FROM Boats B1 NATURAL JOIN Reserves R1, Boats B2 NATURAL JOIN Reserves R2
        - WHERE B1.color='red' AND B2.color='green' AND R1.sid = R2.sid
            - // red boat green boat ensure they're both from the same sailor
    - Same problem (**with set ops**)
        - SELECT R.sid FROM Boats B NATURAL JOIN Reserves R

        - WHERE B.color='red'
        - INTERSECT
        - SELECT R.sid FROM Boats B NATURAL JOIN Reserves R
        - WHERE B.color='green';
    - Find name of sailors who've reserved at least 2 different boats
        - SELECT DISTINCT S1.sname
        - FROM Sailor S1 NATURAL JOIN Reserves R1, Sailor S2 NATURAL JOIN Reserves R2
        - WHERE R1.bid <> R2.bid AND R1.sid = R2.sid
            - // two different boats same sailor
- EXCEPT
    - Subquery1 EXCEPT Subquery2 (same as union and intersection)
        - The two subqueries must be valid queries (SELECT-FROM block w optional WHERE)
    - EXCEPT must be union-compatible (same as union and intersection)
        - The two subqueries must have the same attributes in the SELECT clause
    - (Sometimes MINUS is interchangeable with EXCEPT)
    - EXCEPT excludes duplicates
        - To include duplicates, use EXCEPT ALL
    - Find ID of sailors who've reserved a red boat <u>but never</u> reserved a green boat
        - SELECT sid (dont need to specify since sid is only in Reserves)
        - FROM Boats B NATURAL JOIN Reserves R
        - WHERE B.color='red'
        - EXCEPT
        - SELECT sid
        - FROM Boats B Natural JOIN Reserves R
        - WHERE B.color='green';
    - Find the name of sailors who never reserved a boat. Is this solution correct?
        - SELECT sname
        - FROM Sailors
        - EXCEPT
        - SELECT sname
        - FROM Sailors NATURAL JOIN Reserves;

- **NO** sname is not unique, so one Sam Smith may have reserved, while another Sam Smith hasn't

- Correct Solution:
  - CREATE TABLE Temp_Sid AS
    SELECT sid
    FROM Sailors
    EXCEPT
    SELECT sid
    FROM Sailors NATURAL JOIN Reserves;
  - SELECT sname
  - FROM Temp_Sid NATURAL JOIN Sailors;

10/12/2021

SQL: The Simple Query Language (Part I)

Relational Query Languages

- A major strength of the relational model: supports simple, powerful querying.
- Two sublanguages:
  - DDL -- Data Definition Language
    - Define and modify schema
  - DML -- Data Manipulation Language
    - Queries can be written intuitively.
- DML –Querying Databases
  - A simple SQL query has the form:
    - SELECT A1, A2, ..., An
    - FROM r1, r2, ..., rm
    - WHERE P
      - Ai represents an attribute
      - ri represents a relation
      - P is a predicate
  - The SELECT Clause
    - Equivalent to projection (π) operator
      - $\pi_{\text{sid, sname}}$ Students -> SELECT sid, Sname FROM Students;

        - Can use * to get all attributes (SELECT * FROM Students)
        - By default, duplicates are preserved
        - Can include arithmetic expressions in SELECT
            - Ex: SELECT bname, acct_no, (balance*1.05) FROM account
        - Selecting Distinct Values
            - "SELECT DISTINCT address" to eliminate duplicates
    - The WHERE Clause
        - Equivalent to the selection (σ) operator
            - Only need to use when you need to specify conditions
            - Ex: $\sigma_{\text{course = 'Computing'}}$Students ->
            - SELECT * FROM Students WHERE course = 'Computing';
        - WHERE predicate can be:
            - <attribute> <op> <constant>, or <Attribute1> <op> <Attribute2>
                - op: <, <=, =, **<>**, >=, >
                - **<>: NOT EQUAL -> use this in SQL, use ≠ in Relat. Algebra**
                - In SQL, use AND, OR, NOT
                - In Relat. Algebra, use $\wedge$ (and), $\vee$ (or), and $\neg$ (not)
            - BETWEEN Operator
                - SELECT * FROM Data WHERE index BETWEEN 100 AND 200;
            - IN Operator
                - SELECT Name FROM Member WHERE memno IN (100, 200, 350);
                    - Returns names of members with memno's in the set
                - SELECT Name FROM Member WHERE memno NOT IN (100, 200, 350);
                    - Returns names of members with memno's not in the set
            - IS/NOT Operator
                - SELECT Catno FROM Loan WHERE Date-Returned IS NULL
                    - returns all catno's where date returned is null
                - SELECT Catno FROM Loan WHERE Date-Returned IS NOT NULL
                    - not null -- has a value
            - LIKE Operator

        - '_' stands for any character (length 1)
            - ... LIKE '_EST'; -> Any 4 letter string that ends with EST
        - '%' stands for any string (length >= 0)
            - ... LIKE 'T%'; -> Any string that starts with T
    - Exercise:
        - Schema
            - Boats(<u>bid</u>, name, color)
            - Sailors(<u>sid</u>, sname, rating, age)
            - Reserves(<u>sid</u>, <u>bid</u>, day)
        - Question: Write the WHERE clause of SQL query that finds the boats whose name starts with 'M' and has at least 3 characters
            - WHERE bname LIKE 'M _ _ %';
- The FROM Clause
    - Cross-product (X) or Join (⋈) of tables T1, ...Tn
        - Has multiple tables T1, ... Tn in the FROM clause
    - Distinguish attributes of the same name by "<relation>.<attribute>"
        - E.g., Sailor.name, boat.name
    - **Without WHERE clause**: cross-product (X);
    - **With WHERE clause**:
        - If it checks equivalence on ALL common attributes: natural-join (⋈)
        - The order of tables does NOT matter (the non-joinable tables still can be put side-by-side in FROM class)
        - SELECT A1, ..., An FROM R NATURAL JOIN S;
    - Range Variables
        - Can associate "range variables" with the tables in the FROM clause
            - This saves writing and makes queries easier to understand
        - SELECT x.sname FROM Sailors x, Sailors y WHERE x.age > y.age
            - Returns all but the youngest sailor
    - Exercise
        - Schema:
            - Students (<u>sid</u>, <u>cid</u>, sname, address)

    - Courses (<u>cid</u>, cname)
- R1 = Students ⋈ ($\sigma_{cname='CS442'}$ Courses)
- R2 = $\sigma_{address='Hoboken'}$ R1
- R3 = $\pi_{Students.sname,\ Courses.cname}$ R2
- SQL:

  SELECT cname, sname
  FROM Students NATURAL JOIN Courses
  WHERE cname = 'CS442' AND address = 'Hoboken

10/8/2021

<u>Relational Algebra (Part IV)</u>

In-class Exercises

Division Operator (/)

- Useful for expressing "for all" queries like:
    - *Find SID of sailors who have reserved <u>all</u> boats.*
- Division A/B
    - Put the table that includes "ALL <x>" as the divisor B
    - Attributes of B MUST be a subset of the attributes of A

Aggregate Queries by Relational Algebra

- Relational algebra does not provide any aggregate function in general.
- Relational algebra handles aggregate queries in a complicated way

10/06/2021

<u>Relational Algebra (Part III)</u>

In-class Exercises

10/03/2021

<u>Relational Algebra (Part II)</u>

Compound Operations

- Intersection (∩)
- Join (⋈)
- Division (/)
- Renaming Operation (ρ)

Intersection (AND)

- Notation: R ∩ S
- Output: the tuples in both R and S.
- R and S must be union-compatible.
- ∩ is NOT a basic operation
    - It can be expressed by using basic operations R ∩ S = R - (R - S)
- Is it symmetric?
    - **YES** S1 ∩ S2 = S2 ∩ S1
- Any duplicates?
    - **NO**, S1 ∩ S2 is strictly a subset of S1 and of S2, and S1 and S2 contain no duplicates

Join (⋈)

- Natural Join (⋈)
    - Notation: R ⋈ S
    - Schema: All attributes in R and S
        - Keep only one copy of common attributes in R and S (no R.A and S.A, just one A)
    - Output: All rows in R X S where they have equal values on the common attributes
    - If there are no attributes in common between the two relations, the natural join will return the cross product
- Condition Join ($\bowtie_c$)
    - Notation: R $\bowtie_c$ S
        - c: the condition that the output records must satisfy
    - Condition join R $\bowtie_c$ S is equivalent to σ_c(R X S)
    - Output:
        - Schema: the same as cross-product (common attributes will be represented twice)
            - R.A and S.A will be R.A and S.A, not just A
        - Instances: Only those records in R X S that satisfies condition c
    - Equi-Join
        - A special case of condition join where the condition only contains equalities
        - Result schema: same as cross-product
    - Equi-Join vs Natural Join
        - Equi-Join: the equality operator on specified attributes (not necessarily common)

        - Natural Join: An equi-join on ALL common attributes
- Division (/)
    - Notation: A / B or A ÷ B
    - A/B is used when we wish to express queries with the keyword "ALL":
        - Examples:–"Which students are registered with all the courses taught by Dr. X?"
        - "Which students have taken ALL of the humanities courses?"
    - Output of A/B
        - Attributes of B is proper subset of Attributes of A
        - Attributes of the relation returned by A/B = (All attributes of A - All attributes of B)
            - Ex: Given A(SID,PID,grade) and B(PID), the schema of A/B is (SID,grade)
        - The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple
    - A(SID, Name, Age) and B(SID, Address)
        - Is A/B allowed? **NO**, Address is not in A
        - Is $\pi_{SID, Name}$(A)/π_{SID}(B) allowed? **YES**, SID is in A
        - Is B/π_{SID}(A) allowed? **YES**, π_{SID}(A) is in B
- Renaming Operator (ρ)
    - It allows to name the results of relational-algebra expressions as a new instance
    - Notation: ρ(X, E)
        - Assigns name X to the results of expression E
    - Rename the results of an expression
        - Ex: ρ(OldSailor, $\sigma_{Age > 45}$(Sailor))

10/01/2021

<u>Relational Algebra (Part I)</u>

Relational Query Languages

- Query languages: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
    - Strong formal foundation based on logic.

    - Allows for optimization.
- Query Languages != programming languages
- QLs not intended to be used for complex calculations.
- QLs support easy, efficient access to large data sets.
- Formal Query Languages
    - Relational Algebra: More operational, very useful for representing execution plans.
    - Relational Calculus: Let users describe what they want, rather than how to compute it. (Non-procedural, declarative.)
- Relational Algebra
    - Unary operators (only one table as input)
        - Selection (σ)
        - Projection (π)
    - Set operations
        - Set-difference (-)
        - Union (U)
    - Non-set operation
        - Cross-product operation (X)
    - Unary Operators
        - Projection (π)
            - Pick certain columns (i.e., attributes) for output
            - Notation: $\pi_{A1, A2...Ak}(r)$:
                - r = a relational table
                - A1, ..., Ak: attributes in the output
            - Output returns attributes A_1, ..., A_k in r
            - Any duplicates?
                - **YES**. In non-key attributes duplicate records can be found
        - Selection (σ)
            - Pick certain rows for output
            - Notation: $\sigma_p(r)$
                - r = a relational table
                - p: selection condition

- predicates:
  - <attribute> op <attribute> (ex: salary < bonus), or
  - <attribute> op <constant> (ex: name = 'smith')
- Connectives of predicates: $\wedge$ (and) and $\vee$ (or)
  - ex: name = 'Smith' $\wedge$ age = 20

- Schema of result: the same as the schema of the input relation r
- Instance:
  - The records that satisfy the selection condition
  - Question: does the selection results contain duplicates?
    - **NO**. It is a subset of the table, and the table has a key attribute, so the subset will always have a key and thus always a unique column in it

- Order between Selection (σ) and Projection (π)
  - Projection π is ALWAYS placed before selection σ.
  - Format (put σ closest to r, and π farthest from r)
    - $\pi_{A1, A2...Ak}\ \sigma_p(r)$

- Set Operators
  - Union (U)
    - Notation: R U S
    - Returns a relation instance containing all tuples that are in either R or S (or both)
    - Symmetry of U: R U S = S U R
  - Set-difference (-)
    - Notation: R - S
    - Returns a relation instance containing all tuples that are in R AND not in S
    - Ever any duplicates?
      - **NO**. R - S is always a subset of R
    - Symmetric? R - S == S - R?
      - **NO**. R - S will only be a subset of R, S - R will only be a subset of S. Subset of R could not be equivalent to subset of S
- Non-set Operators

- Cross-Product (X) Operator
  - Notation: R X S
  - Schema of output: the attributes of R followed by the attributes of S, in order.
    - If R and S contain the same attribute A, the output schema includes both R.A and S.A
  - Instances in the output:
    - A Cartesian product of R and S
    - Pair each tuple of R with each tuple of S
  - Any duplicates?
    - **NO**. Will always be (r_i,s_j) where R contains no duplicates and S contains no duplicates, so each pair will be unique
    - Also in notes

9/29/2021

ER to Relational Model Mapping (Part II)

All in notes

9/27/2021

ER to Relational Model Mapping (Part I)

Rules

- For each entity set: construct a table
- For each relationship set: construct a table if necessary (more on this later)
- For each table, decide its schema:
  - Decide the attributes
  - Decide primary key, candidate keys, and foreign keys

Translate ER Diagrams into Relational Tables

- Strong Entity Set (in notes)
- Weak Entity Set (in notes)
  - Foreign key: $\underline{a_1}$
  - Never a table for a relationship in **weak -> relation** - strong

    - Key of **weak** is a composite key (a1,b1)
- Relationship Sets
    - Translate a Many-to-Many relationship set R to a relation table
        - Attributes: $<c_1 \dots c_k> + \underline{a_1} + \underline{b_1}$ (primary keys of E1 and E2)
        - Primary Key: (a_1,b_1) -- a composite key
        - Two foreign keys
            - $\underline{a_1}$ references E1
            - $\underline{b_1}$ references E2
        - General example in notes
    - 1-to-Many Relationship set R
        - Attributes (same as m:n)
            - All attributes of the relationship set
            - The primary key of each participating entity set
        - Primary Key (diff)
            - Only contain the key attributes of the entity set at the **Many side**
        - Foreign Key (same as m:n)
            - Define a foreign key for each participating entity set

9/24/2021

Relational Instance: Don't allow duplicate records or non-atomic (non-singular valued) cells, keys cannot contain null values, duplicate attribute names are unacceptable

9/22/2021

The Relational Model (Part III)

Foreign Key

- A foreign key of relation R is a set of attributes that is a key of relation S (S is different from R)
- FK enforces the referential integrity that some attributes in R must REFER to the key of S

```
CREATE TABLE <name> (
        <field1> <domain>,
        <field2> <domain>,
        ...
        PRIMARY KEY (field1, field2,...),
        UNIQUE (field1, field2,...),
```

FOREIGN KEY (<List of attributes>) REFERENCES Table1 (<List of attributes>)

)

Notes:

- Referenced attributes in REFERENCES statement must be declared PRIMARY KEY or UNIQUE in Table1.
- Reference attributes can be omitted if they have the same names as foreign keys.
- Unlike a primary key, a foreign key need not be unique (i.e., a foreign key is not necessarily a key).

<u>To Define a Foreign Key:</u>

CREATE TABLE ENROLLED (

sid CHAR(20),

cid CHAR(20),

grade float,

PRIMARY KEY (sid,cid),

FOREIGN KEY (sid) REFERENCES Students);

<u>Reference Options for DELETE/UPDATE</u>

CREATE TABLE <name> (

...

FOREIGN KEY (<list of attributes>)

REFERENCES Table1 (<list of attributes>)

[ON DELETE reference_option]

[ON UPDATE reference_option];

Note: reference_option = {CASCADE, NO ACTION, SET NULL, SET DEFAULT}

- Cascade: Delete/update automatically
- No Action: Do nothing
    - An error is raised, or SQL statement is rolled back
- Set Null: Set referenced attributes to NULL
    - Can be specified only if some column of the foreign key allows NULL values
- Set Default: Set referenced attributes to their default

Referenced Attributes != Foreign Key, Foreign Key MUST be primary key of new table

9/20/2021

The Relational Model (Part II)

Integrity Constraints

- Nice Properties of Keys
    - Given a set of attributes A (can contain 1 or multiple attributes)
        - If A is a key, then any superset A' of A CANNOT be a key (key must be minimal)
    - If A is not a key, then any subset A' of A CANNOT be a key
        - If (Name,Birthdate) is not a key, then neither (Name) nor (Birthdate) is a key
- Find all Keys Efficiently
    - Construct an *attribute lattice* (notes) to enumerate all possible combinations of attributes
        - Top of lattice: each node contains a single attribute
        - Bottom of lattice: one node that contains all attributes
        - Edge N1->N2: N1 is a strict subset of N2

Define Primary Keys by SQL

- SQL Syntax for Primary Key definition:
  CREATE TABLE <name> (<field1> <domain>, <field2> <domain>, ..., PRIMARY KEY (<attributes>));
- Candidate Keys:
  CREATE TABLE <name> (... PRIMARY KEY(...), UNIQUE (<attributes>), UNIQUE(<attributes>));
- **For Singleton Keys**: Can put PRIMARY KEY or UNIQUE after the type (domain) in the declaration

Foreign Keys

- Referential Integrity Constraint: Any enrolled student must be a student first (notes).
    - Enrolled students that have no matching record as students is a bad thing, so reject that data entry.
    - How to deal with invalid data entry?
        - Remove all Enrolled tuples that refer to it
        - Disallow deletion of a Students tuple that is referred to
        - Set the sid in the Enrolled tuples that refer to it to a *default sid*
        - Set the sid in the Enrolled tuples that refer to it to a NULL value

9/17/2021

The Relational Model (Part I)

Relational Database

- A set of relations
- Made up of two parts:
    - Schema: Specifies the name and attributes of relations
        - MUST be defined first
    - Instance: A table with rows and columns, consistent with schema
- Relational Schema
    - Usage: relation_name(attr1: type, ..., attrN: type)
    - Students(sid: string, name: string, login: string, age: integer, GPA: real)
    - Attributes are referenced by name, not column locations, MUST be unique
- Relational Instances
    - Instance: A table with rows and columns
    - Attributes (or fields) are stored in columns
    - Tuples (or records) are stored in rows
    - Attributes have a domain - an atomic type
    - Cardinality of a relation R: Number of rows in R (excluding title row)
    - Degree/arity of a relation R: Number of columns in R
- Notes of Relational Model
    - No duplicate tuples in a relation
        - What if we want to insert duplicate tuples? What can we do?
            - Add an index attribute!
    - Ordering
        - No ordering of tuples in a relation
    - The value of each attribute is either drawn from its domain or the special value NULL
    - Attribute's values are atomic

SQL: Standard Language to describe and manipulate relational database

- Data Definition Language (DDL)
    - Create, modify, delete relations
    - Specify constraints
    - Administer users, security, etc.
- Data Manipulation Language (DML)
    - Specify queries to find tuples that satisfy criteria

    - Add, modify, remove tuples
- Data Types
    - All attributes MUST have a data type
    - **Character**: CHAR(n): fixed-length string of n characters
        - VARCHAR(n): a set of characters up to length n
    - **Boolean**: An attribute whos value is logical (TRUE, FALSE, UNKNOWN)
    - **Date & Time**: Represents dates and times
    - **INT or INTEGER**: denotes an integer
    - **FLOAT**: denotes floating-point numbers
        - DECIMAL(n,d) allows values that consist of n decimal digits, where the decimal point is d positions from the right
    - Ex:

      CREATE TABLE Enrolled(sid: CHAR(8), cid: CHAR(8), grade: FLOAT);
- Case-sensitivity of Table/Column Names
    - By default, case-sensitive on Linux, insensitive on Windows
    - MySQL can configure it to be either way
- Creating Instance in SQL
    - Syntax:

      INSERT INTO <TableName> (<field names>)

      VALUES(<field values>)

      Ex -- Insert a single tuple:

      INSERT INTO Students (SID, Name, Login, Age, GPA)

      VALUES('53688', 'Smith', 'smith@ee', 18, 3.2);

      Ex -- Insert multiple tuples:

      INSERT INTO Students (SID, Name, Login, Age, GPA)

      VALUES('53688', 'Smith', 'smith@ee', 18, 3.2),

      VALUES('53701', 'Philth', 'philth@ee', 19, 3.4);

Integrity Constraints (ICs)

- Conditions specified on a database schema
- Key of Relational Instances
    - Key: The minimal subset of the attributes that uniquely identifies a tuple in the instance

        - The singleton key only contains 1 attribute
            - Ex: (SSN)
        - The composite key contains multiple attributes; these attributes together identifies each tuple uniquely.
            - Ex: (Name, BirthDate)
        - An instance can have multiple keys
            - Ex: The student table can have 3 keys: (SSN), (Name, BirthDate), and (Name, DormRoomNumber)
        - Some composite keys can overlap
            - Ex: (Name, Birthdate), (Name, DormRoomNumber)
            - Singletons can NEVER overlap with composites
    - Superkeys & Primary Keys
        - Similar to superkeys and primary keys in ER diagram
        - A set of fields is a superkey if no two distinct tuples have the same values in these key fields
            - ALL attributes of a relation together form a super key for the relation
        - >1 key for a relation?
            - One of the keys is chosen (by DBA) to be the primary key.
            - The other keys are called candidate keys.
            - All the keys (primary and candidate keys) of a relation cannot contain a NULL value as a value for their components
                - Composite keys (A,B) mean neither column A nor B contains any NULL values

9/15/2021

Exercises

1) University DB contains info about professors (by SSN) and courses (by courseID).

2) Design a small database for an on-line trading site (such as Amazon) that sells used textbooks.

- Each textbook has a unique ISBN, a category (e.g., Math) it belongs to, and a name. In addition, each book has a status description.

- Each textbook is published by a publisher. Each publisher has to have published at least one textbook. The same textbook can be published by multiple publishers, while each publisher can publish multiple textbooks.
- For each publisher, the database maintains its unique name, address, phone numbers (could be more than one phone number, each with a number), and its contact person (who is an individual with all the related information for individuals, see next sentence). There is only one contact person for each publisher.
- No two publishers can have the same phone number.
- For each individual contact person, the database keeps a name, a phone number and a unique email address. An individual can be the contact person of multiple publishers.
- Each textbook has one or more sellers, which may be either companies (corporation sellers) or individuals (individual sellers). Each seller has a name, a unique online ID, and rating. Each seller can sell one or more textbooks.
- Each corporate seller has a business address, and each individual seller has a "Ship from" address. Draw the ER diagram based on the facts.

9/13/2021

<u>Continued Issues for ER Diagram design:</u>

- Entity Set vs Relationship Set
    - At semantic level: If a concept describes the association between multiple entity sets, define it as a relationship set. Otherwise, define it as an entity set
    - Be careful of where to put attributes
        - If attribute A is associated with the entity set E, each instance of E only has one value of A
        - If attribute A is associated with the relationship set R(E, E')
            - Each instance of R(E,E') only has one value of A
            - Each instance of E can be associated with multiple values of A
- Binary vs Ternary Relationships
    - Are binary relationships always better than ternary relationships?
    - Example: Entities: Person, Bar, Beers
        - Design the relationships that record which person drank which beers at which bar and when

- Design Clean, Succinct ER Diagrams
    - Avoid Redundancy
        - Saying the same thing in two (or more) different ways
        - Wastes space and (more importantly) encourages inconsistency
    - Don't use an entity set when an attribute will do

9/10/2021

<u>Advanced Issues for ER Diagram design:</u>

- Hierarchy
    - Subclasses = special case = fewer entities = more properties
        - Ex: Ales are a kind of beer. Not every beer is an ale, but some are
        - Ales have all the properties of beer + new attribute 'color'
    - One entity type might be a subtype of another
        - Ales is a subtype of beer
        - Beer is a supertype of Ales
    - The "is a" relationship exists between the superclass entity and its subclass entity
        - Ales IsA beer
        - Drawn as a triangle and is always "IsA"
    - Properties of IsA relationship
        - Inheritance: Subclass inherits all attributes of superclass
            - The key of the superclass is the key of the subclass
            - Subtypes can have new attributes
        - Transitivity
            - Ales is a subclass of Beer; Beer is a subclass of Alcoholic Drink, so Ales is also a subclass of Alcoholic Drink
        - Overlap Constraints: Some instances can appear in two or more subclass entity sets
        - Covering Constraints: All instances in the superclass entity set are covered by at least one subclass entity set
- Aggregation
    - Motivation Example
        - How to model relationships between relationships?

            - Draw a big box around entity sets that are directly connected to the relationship that is related to another relationship
                - Attributes of directly connected entity sets are included inside the aggregation box
        - Can we merge the related relationships into one?
            - NO, each relationship is distinct, with descriptive attributes, and cannot be merged into one.
- Design Issues of ER Diagram
    - Conceptual Design Using the ER Model
        - Design Choices
            - Should a concept be modeled as an entity set or an attribute?
            - Should a concept be modeled as an entity set or a relationship?
            - Identifying relationships: Binary or ternary or more?
    - Design Issue #1: Entity Set vs Attribute
        - Sometimes a concept can be represented as either an entity set or attributes
        - A concept C should be defined as an entity set if one or more of the following conditions are satisfied:
            - It is more than the name of something; it has at least one non-key attribute
            - More than one instance of C can be associated with one single entity
            - More than one instance of C can be associated with one single relationship
        - Rule #1: More than the name of something; it has at least one non-key attribute
            - Ex: If semester is associated with other information
        - Rule #2: More than one instance of C can be associated with one single entity
            - Ex: Student can have both home address and school address
        - Rule #3: More than one instance of C can be associated with one single relationship
            - Ex: Student can take the course in different semesters
            - A (student, semester) pair can be associated with multiple semesters
        - Ex: Each person has his/her contact info as a mobile phone number -> ATTRIBUTE
            - (Assuming) each person only has one mobile phone

- Each person has his/her phone number as work and as mobile -> ENTITY SET

9/8/2021

The Entity-Relationship Model (III)

Cardinality Constraints

- In 1:M, M:1, the arrow must ALWAYS point from the Many to the relationship.

Participation Constraint

- An entity set E may participate in a relation R either totally or partially
  - **Total Participation**: Every entity in E is involved in the relationship R
    - Represented by a **BOLD LINE** between entity and relationship set
  - **Partial Participation**: Not all entities in E are involved in the relationship R
    - Represented by a LINE between entity and relationship set

Strong and Weak Entities

- Strong Entity: Entities that exist independently
- Weak Entity: Entity that depends on other entities
  - Weak Entity Set: An entity set cannot exist by itself. Its entities owe their existence to some entity in a strong entity set.
  - REQUIRED ACTIONS in S - R - W
    i. Weak entity set W: draw the **rectangle** in **bold line**
    ii. Identifying relationship R: draw **diamond** in **bold line**
    iii. Draw a **bold line** connecting **R** and **W** (total participation)
    iv. Add an arrow on the line that connects R and W, with the arrow pointing to R (one-to-many relationship)
  - Weak entity set never has a key, it only has a partial key
    i. Partial Key represented by dashed underline - - - - - -
    ii. Key entity of weak entity set = primary key of owner entity + partial key of itself

9/3/2021

The Entity-Relationship Model (II)

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Ellipses represent attributes

- Underline indicates primary key attributes

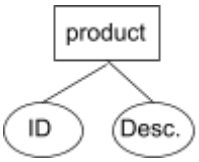
9/1/2021

The Entity-Relationship Model (I)

Steps in Database Design

1. Requirements Analysis -- user needs (non-professional clients)
2. Conceptual Design -- high-level description
3. Logical Design -- translate ER into a DBMS data model
4. Schema Refinement -- remove redundancy, normalization
    a. this is what makes a GOOD design for a database

ER Model

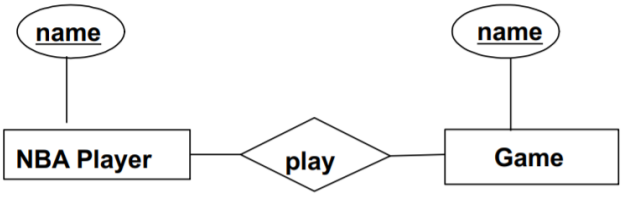
- E: Entities, R: Relationships
- Entity Set
    - Entity: Real-world object, distinguishable from other objects, described using a set of attributes
    - Set: Collection of similar entities, typically by grouping some attribute
        - Each entity must have a unique value (key) to identify them
    - In diagram: Entities -> rectangle, their attributes -> circle, key -> underlined
- Ex: ShopRite hires you to design database for product inventory. Each product has an ID, description, category, price, expiration date
    - 

Three Types of Keys

- Super Key
    - Set of attributes which, collectively, uniquely identify an entity in an entity set
    - Ex: Using ID, Desc, etc., together as key for a product (from the diagram above), doesn't need to contain all attributes, just a subset
- Key
    - A superkey for which no proper subset is a superkey (~ minimal superkey)

    - Each entity set must have at least one key, key can have more than one attribute if each individual attribute cannot uniquely identify each entity
- Primary Key
    - The key that is chosen as the principal means to identify entities in an entity set
    - Since there can be more than one key for an entity set, the primary key is the one that is underlined in ER diagram as the primary means of identification
    - If primary key, then key. If key then superkey.

Relationship Set

- Relationship: Association among two or more entities
- Degree of a Relationship
    - Degree: Number of participating entities
        - 2 -> binary, 3 -> ternary, n -> n-ary
- Example: NBA players playing games
    - 

- Attributes with Entities/Attributes with Relationships (adding year)
    - With Entity: Each Game Entity is associated with a single value of ‘year’
    - With Relationship: Each (Player,Game) pair is associated with a single value of ‘year’
        - With relationship allows the same player to play the same game in different years
        - With entity cannot