CS4211
AY24/25 SEM 1

01. THEORY

Keys

- **Superkey** $\rightarrow$ A subset of attributes that uniquely identifies a tuple (all attributes) in a relation.
- **Key** $\rightarrow$ A minimal **superkey**. (Cannot be made smaller)
- **Candidate Keys** $\rightarrow$ The set of all **keys** of a given relation.
- **Primary Key** $\rightarrow$ The selected **candidate key**.
- **Foreign Key** $\rightarrow$ A subset of attributes of relation R_1 that refers to the **primary key** of relation R_2. It can contain `NULL` values.

02. ER DIAGRAMS

Cardinality Constraints

- Example item

Participation Constraints

- Example item

ISA Hierachies

Aggregation

03. SQL

Three-valued logic

- SQL uses `True`, `False` and `NULL`.
- Suppose x is `NULL`
  1. `x IS NULL` $\rightarrow$ `True`
  2. `x IS NOT NULL` $\rightarrow$ `False`
  3. `x IS DISTINCT FROM y`
     - y is `NULL` $\rightarrow$ `False`
     - y is **not** `NULL` $\rightarrow$ `True`
  4. `x IS NOT DISTINCT FROM y`

Note: `NULL <> NULL` $\rightarrow$ `NULL`

Constraints

Unique $\rightarrow$ `r1.column <> r2.column`

Named constraints: `id INT CONSTRAINT nn_id NOT NULL`

Foreign Key Constraints: `ON DELETE <ACTION> / ON UPDATE <ACTION>`

- `NO ACTION`: Reject if it violates constraints
- `RESTRICT`: Can't be deferred
- `CASCADE`
- `SET DEFAULT`
- `SET NULL`

Check Constraint

- Single Column: `CHECK(hours > 0)`
- Multiple Columns: `CHECK(start_year <= end_year)`

Pattern Matching

- `_` matches any single character.
- `%` matches a sequence of 0 or more characters.

Set Operations

Union Compatible $\rightarrow$ Two relations R and S are **union compatible**

1. R and S have the same number of attributes
2. The corresponding attributes have the same data type / compatible domains.

- `Q1 UNION Q2` $\Longrightarrow Q_1 \cup Q_2$
- `Q1 INTERSECT Q2` $\Longrightarrow Q_1 \cap Q_2$
- `Q1 EXCEPT Q2` $\Longrightarrow Q_1 - Q_2$

SQL Conceptual Evaluation

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `SELECT`
6. `ORDER BY`
7. `LIMIT / OFFSET`

04. RELATIONAL ALGEBRA

Unary Operators

- **Selection** $\sigma_{[c]}(R)$ Selects all tuples from a relation R that satisfy the condition c based on the **principle of acceptance**.
- **Projection** $\pi_{[\ell]}(R)$ Keeps only the columns specified in the ordered list ℓ and in the same order.
- **Renaming** $\rho_{[\mathbb{R}]}(R)$ Rename all the attributes mentioned in $\mathbb{R}$ such that for each renaming $B_i \leftarrow A_i$, the attribute A_i is renamed to B_i.

Binary Operators

- **Union** $\cup$
- **Intersect** $\cap$
- **Except** $-$
- **Cross Product** $\times$

SQL in RA

$$\pi_{[attributes]}(\sigma_{[condition]}(R_1 \times R_2))$$

- π Projection $\rightarrow$ `SELECT`
- $\times$ Cross Product $\rightarrow$ `FROM`
- σ Selection $\rightarrow$ `WHERE`

Joins

- **Inner Join** $\bowtie_{[\theta]}$
- **Equi Join** $\bowtie_{[=]}$
- **Natural Join** $\bowtie$ If there is no common attributes then it is a cartesian join.
- **Left Outer Join**
- **Right Outer Join**
- **Full Outer Join**

05. FUNCTIONS, PROCEDURES, TRIGGERS

Functions

- Function parameters can also serve as variables using the `IN` and `OUT` keyword, for example `FUNCTION swap(INOUT val1 INT, INOUT val2 INT)` and `FUNCTION sum_to_x(IN x INT, OUT s INT)`
- The `return_type` can be an atomic value like `INTEGER`, `CHAR(10)` or a tuple such as `RECORD` or other table's names like `Scores` or even tables like `TABLE(name TEXT, mark INT, gap INT)`

```
CREATE OR REPLACE FUNCTION func_name(val INT)
RETURNS return_type AS $$
  DECLARE
  -- local variables
  BEGIN
  -- implementation
  END;
$$ LANGUAGE plpgsql;
```

Procedures

- Procedures have no return type.
- Example procedure usage: `CALL transfer('Alice', 'Bob', 100)`

```
CREATE OR REPLACE PROCEDURE p_name(val INT)
AS $$
  DECLARE
  -- local variables
  BEGIN
  -- implementation
  END;
$$ LANGUAGE plpgsql;
```

Control Structures

- `IF... THEN... ELSE... ENDIF`
- `LOOP... END LOOP`
- `EXIT... WHEN...`
- `WHILE... LOOP... END LOOP`
- `FOR... IN... LOOP... END LOOP`

Cursors

Cursors enable us to access each individual row returned by a `SELECT` statement.
Flow: Declare $\rightarrow$ Open $\rightarrow$ Fetch $\rightarrow$ Not Found $\rightarrow$ Close

Cursor Movement

- `FETCH curs INTO r` $\rightarrow$ Next tuple
- `FETCH PRIOR FROM curs INTO r` $\rightarrow$ Prior tuple
- `FETCH FIRST FROM curs INTO r` $\rightarrow$ First tuple
- `FETCH LAST FROM curs INTO r` $\rightarrow$ Last tuple
- `FETCH ABSOLUTE 3 FROM curs INTO r` $\rightarrow$ 3rd tuple

Triggers

```
CREATE OR REPLACE TRIGGER func_name()
RETURNS TRIGGER AS $$
  -- implementation
$$ LANGUAGE plpgsql;

CREATE TRIGGER foo_trigger AFTER INSERT ON Table
FOR EACH ROW EXECUTE FUNCTION func_name();
```

NEW and OLD keywords

- `NEW`: Insert and Update
- `OLD`: Update and Delete

Before and After keywords

- `BEFORE`: Executed before the `TG_OP`
- `AFTER`: Executed after the `TG_OP`
- `INSTEAD OF`: Trigger function is executed instead of `TG_OP`

Row and Statement level Triggers

- `FOR EACH ROW`: Execute trigger function for every tuple encountered.
- `FOR EACH STATEMENT`: Execute trigger function once per statement.

Order of Trigger Activation

1. Before Statement Triggers
2. Before Row Triggers
3. After Row Triggers
4. After Statement Triggers

Deferred Trigger

```
CREATE CONSTRAINT TRIGGER bal_check_trigger
AFTER INSERT OR UPDATE OR DELETE ON Account
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW EXECUTE FUNCTION bal_check_func();
```

- Deferred triggers only work for `AFTER` and `FOR EACH ROW`.
- Use `BEGIN TRANSACTION` and `COMMIT` to defer a deferrable trigger.

06. NORMALIZATION

Functional Dependency

- $\{A\} \to \{B\}$ denotes that attribute A uniquely decides another attribute B.
- Two objects that have the **same** A value should have the **same** B value.

Determining FDs from requirements

- Suppose we have a requirement *no two shops should sell the **same product** to the **same customer** at the **same day** at **two different prices***.
- Identify the attributes involved: *ShopID*, *ProductID*, *CustomerID*, *Date* and *Price*.
- Create a tuple that violates the requirements.

ShopID	ProductID	CustomerID	Date	Price
S_1	P_1	C_1	16/4/2024	P_1
S_1	P_1	C_1	16/4/2024	P_2

- The attribute that is **different** in this case the *price* should be uniquely decided by the attributes that remain the **same** which are *ShopID*, *ProductID*, *CustomerID* and *Date*.

Armstrong's Axioms

1. **Reflexivity:** $\{A, B\} \to \{A\}$
2. **Augmentation:** $\{A\} \to \{B\} \implies \{A, C\} \to \{B, C\}$
3. **Transitivity:** $\{A\} \to \{B\}$ and $\{B\} \to \{C\} \implies \{A\} \to \{C\}$

Additional Rules

- **Rule of Decomposition:**
  $\{A\} \to \{B, C\} \implies \{A\} \to \{B\}$ and $\{A\} \to \{C\}$
- **Rule of Union:** $\{A\} \to \{B\}$ and $\{A\} \to \{C\} \implies \{A\} \to \{B, C\}$

Computing Closures

Given $\{A_1, A_2, \ldots A_n\}$, the closure $\{A_1, A_2, \ldots A_n\}^+$ can be computed by:

1. Initialize the closure to $\{A_1, A_2, \ldots A_n\}$.
2. If there is a FD $A_i, A_j, \ldots A_m \to B$ such that $A_i, A_j, \ldots A_m$ is in the closure, then put B in the closure.
3. Repeat step 2 until we cannot find any new attribute to put in the closure.

Keys, Superkeys, Prime Attributes

- **Superkey:** $\to$A set of attribute(s) in a table that decides all other attributes.
- **Key:** $\to$A superkey that is minimal.
- **Prime Attribute:** $\to$If an attribute appears as part of a key, it is a prime attribute.

Example: $\{A, B\}$ and $\{D\}$ are keys for $R(A, B, C, D)$ then the prime attributes for the table are A, B and D.

Tricks to find keys

1. Check small attribute sets first.
2. If an attribute A_i does not appear on the RHS of any FD then A_i must be part of a key.

BCNF Definitions

- **Decomposed FD:** $\to$An FD whose *right hand side* has only **one attribute**
- **Trivial FD:** $\to$An FD whose attribute(s) on the *RHS* appears on the *LHS*.
- **Non-trivial FD:** $\to$An FD whose attribute(s) on *RHS* **do not appear** on *LHS*.

BCNF

A table R is in BCNF if **every non-trivial and decomposed FD** has a **superkey** on its *left hand side*.

BCNF Simplified Check

Compute the closure of each attribute subset in R and check if there exists a **more but not all** closure. If such a closure exists then R is not in BCNF.

BCNF Decomposition

Input: A table R

1. Find a subset X of attributes in R such that its closure $\{X\}^+$ contains **more attributes** than X but **does not contain all attributes** in R.
2. Decompose R into two tables R_1 and R_2, such that
   - R_1 contain all attributes in $\{X\}^+$
   - R_2 contains all attributes in X as well as the attributes not in $\{X\}^+$.
3. If R_1 is not in BCNF, further decompose R_1.
4. If R_2 s not in BCNF, further decompose R_2.

Notes:

- BCNF decomposition is a binary decomposition.
- The BCNF decomposition of a table may not be unique.
- If a table only has two attributes then it is in BCNF.

Checking if a Decomposed Table is in BCNF

- Suppose we decomposed a table R into R_1 and R_2. If we want to check of the decomposition δ is in BCNF, then we need to individually check that both R_1 and R_2 are in BCNF.
- Here is how to check if one decomposed table R_i is in BCNF.
  1. Enumerate the attribute subsets of R_i.
  2. Derive the closures of these attribute subsets on R,
  3. Project these closures onto R_i by removing attributes not in R_i.

Lossless Join Decomposition

- When we decompose a table R into R_1 and R_2, it is a **lossless-join decomposition** if the common attributes $R_1 \cap R_2$ constitute a **superkey** for either R_1 or R_2.
- To determine if the relations R_i, R_j in a decomposition δ is a **lossless-join decomposition**, give an example, where for all $R_i, R_j \in \delta$, $R_i \cap R_j$ constitute a **superkey** for either R_i or R_j.
- BCNF guarantees lossless-join decomposition.

3NF

A table satisifies 3NF if and only if for every non-trivial and decomposed FD, either the **left hand side is a superkey** or the **right hand side is a prime attribute**.

Dependency Preservation

- Let S be the given set of FDs on the original table.
- Let S' be the given set of FDs on the decomposed tables.
- The decomposition preserves all FDs if and only if S and S' are equivalent.

Dependency Preserving Decomposition

Given a set of FDs $\sum$ and decomposition δ:

1. Enumerate the attribute subsets of each decomposed table R_i in δ.
2. Derive the closures of these attribute subsets on R,
3. Project these closures onto R_i by removing attributes not in R_i.
4. Convert the closures in step 3 into non-trivial FDs and combine all the FDs for each R_i into a set S.
5. Check if $\sum$ and S are equivalent, if not identify the FDs that are not preserved.

Minimal Basis

- Let S be a set of FDs
- The minimal basis M is a set of FDs such that
  1. Every FD in S can be derived from M and vice versa.
  2. Every FD in M is a non-trivial and decomposed FD.
  3. If any FD is removed from M, then some FD in S cannot be derived from M.
  4. For any FD in M, if we remove an attribute from its left hand side, the FD cannot be derived by S.
- **Minimal Basis Algorithm**
  1. Transform the FDs such that each RHS **only contains one attribute.**
  2. Examine FDs which have more than one attribute on the LHS and remove redundant attributes on the LHS.
  3. Remove redundant FDs.
     - Pretending to remove one FD at a time from the remaining FDs in S. Compute the closure of the LHS and check if the RHS is still in the closure.
     - Check for transitivity in the existing FDs.

A **canonical cover** is formed by using the rule of union on FDs in the minimal basis that have the same LHS.

3NF Decomposition

Input: A table R with a set of FDs.

1. Find a minimal basis of the FDs.
2. Combine the FDs whose left hand sides are the same.
3. For each FD, construct a table that contains all atributes in the FD.
4. Check if any table contains a key for R. If not create a table that contains a key for R.
5. Remove subsumed tables.