CS2102 Cheatsheet:Database Systems by randomwish

ER Diagram Basics

Core Components

- **Entity Sets:** Represented as rectangles. Hold objects of the same type (e.g., `students`).
- **Attributes:** Represented as ellipses. Can be:
  - **Simple:** Atomic values (e.g., `name`).
  - **Composite:** Made of sub-parts (e.g., `name = (first, last)`).
  - **Derived:** Computed from other attributes (e.g., `age` from `dob`).
  - **Multivalued:** Set of values (e.g., `phones`).
- **Primary Key:** Underlined attributes uniquely identifying entities.

Relationships

- **Relationships:** Represented as diamonds. Connect entity sets.
- **Participation:**
  - **Total:** Every entity in the set participates (bold line).
  - **Partial:** Optional participation (regular line).
- **Cardinality:** Specifies how entities are related:
  - **1:1:** One-to-one - tables may be merged
  - **1:N:** One-to-many - Use a primary key that includes the foreign key from the "many" side.
  - **M:N:** Many-to-many.

Special Constructs

- **Weak Entities: Filled-in dot from its primary key to an entity of its "dominating" set** Cannot exist without a strong entity; linked via an identifying relationship (Diamond between the two sets, with attributes).
- **Aggregation** (Rectangle in a Diamond): Used when a relationship set participates in another relationship.

Key SQL Concepts

Basics

- `SELECT`: Retrieve specific columns.
- `WHERE`: Filter rows based on conditions.
- `GROUP BY`: Aggregate results into groups.
- `HAVING`: Filter aggregated results.
- `ORDER BY`: Sort rows (`ASC`, `DESC`).

Joins

- `INNER JOIN`: Matches rows based on a condition.
- `LEFT JOIN`: Includes unmatched rows from the left table.
- `RIGHT JOIN`: Includes unmatched rows from the right table.
- `FULL JOIN`: Includes unmatched rows from both tables.

Set Operations

- `UNION`: Combine rows, removing duplicates.
- `INTERSECT`: Common rows in both queries.
- `EXCEPT`: Rows in the first query but not the second.

Nested Queries

- **WHERE Clause:** Filter rows using subqueries.
- **FROM Clause:** Use subqueries as derived tables.
- **EXISTS:** Check if a subquery returns any rows.
- **NOT EXISTS:** Check if a subquery DOES NOT return any rows.
- **ANY/ALL:** Compare a value to subquery results.

Programming with SQL

Stored Functions

Encapsulate SQL logic for reuse. Supports conditionals and loops.
Example: Compute factorial of n:

```
CREATE FUNCTION factorial(n INT) RETURNS INT AS $$
DECLARE
    result INT := 1;
BEGIN
    FOR i IN 1..n LOOP
        result := result * i;
    END LOOP;
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Cursors

Process rows one by one. Useful for large datasets.
Example: Cursor to calculate total points:

```
DECLARE c CURSOR FOR SELECT points FROM students;
DECLARE total INT := 0;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO total_points;
        EXIT WHEN NOT FOUND;
        total := total + total_points;
    END LOOP;
    CLOSE c;
END;
```

Triggers in Database Systems

Overview of Triggers

- **Triggers**: Event-Condition-Action (ECA) rules in a database.
  - **Event**: Specifies when the trigger is activated (e.g., `INSERT`, `UPDATE`, `DELETE`).
  - **Condition**: Boolean test that must be satisfied.
  - **Action**: Operation performed if the condition is true.
- Built on stored functions.
- Common use cases:
  - Maintain data integrity.
  - Automate logging.
  - Handle constraints beyond schema capabilities.

Trigger Options

- **Events**: `INSERT`, `UPDATE`, `DELETE`, and combinations.
- **Timing**:
  - `BEFORE`: Trigger fires before the operation.
  - `AFTER`: Trigger fires after the operation completes.
  - `INSTEAD OF`: Trigger fires in place of the operation (used for views).
- **Granularity**:
  - `FOR EACH ROW`: Trigger executes for each affected row.
  - `FOR EACH STATEMENT`: Trigger executes once per statement.

Trigger Refinements

- **Conditions**: Boolean expressions in trigger definitions (e.g., `WHEN (NEW.value <> OLD.value)`).
- **Deferrable Triggers**:
  - `INITIALLY DEFERRED`: Executes at the end of a transaction.
  - Useful to handle constraints spanning multiple operations.

Example: Logging Changes to Student Points

Use Case

- **Goal**: Store the type of operation (`INSERT`, `UPDATE`, or `DELETE`), the old points, and the new points in a log table.

Implementation

```
-- Table to log point changes
CREATE TABLE points_log (
    student_id INT,
    operation TEXT,
    points_old INT,
    points_new INT,
    created_at TIMESTAMP DEFAULT NOW()
);

-- Trigger function to log changes
CREATE FUNCTION log_student_points() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO points_log (student_id, operation, points_old,
        VALUES (NEW.id, TG_OP, NULL, NEW.points, DEFAULT);
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO points_log (student_id, operation, points_old,
        VALUES (OLD.id, TG_OP, OLD.points, NULL, DEFAULT);
    ELSIF TG_OP = 'UPDATE' THEN
        IF NEW.points <> OLD.points THEN
            INSERT INTO points_log (student_id,
            operation, points_old, points_new)
            VALUES (OLD.id, TG_OP, OLD.points,
            NEW.points, DEFAULT);
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger to log changes
CREATE TRIGGER on_student_points_change
AFTER INSERT OR DELETE OR UPDATE ON students
FOR EACH ROW
EXECUTE FUNCTION log_student_points();
```

Final Notes

- Triggers execute in the order:
  1. `BEFORE` statement-level.
  2. `BEFORE` row-level.
  3. `AFTER` row-level.
  4. `AFTER` statement-level.
- Caution with recursive or circular triggers as they can cause infinite loops.

Relational Algebra

- **Projection (π):** Select specific attributes.
- **Selection (σ):** Filter rows by condition.
- **Union ($\cup$):** Combine rows.
- **Difference ($-$):** Rows in one relation but not the other.
- **Join ($\bowtie$):** Combine related rows from two relations.
- **Cartesian Product ($\times$):** Combine all rows from two relations.

Functional Dependencies (FDs)

Definitions

- **FD:** $X \to Y$, $X, Y \subseteq R$, means if two tuples agree on X, they agree on Y.
- **Trivial FD:** $X \to Y$ is trivial if $Y \subseteq X$.
- **Non-Trivial FD:** $X \to Y$ is non-trivial if $Y \nsubseteq X$.
- **Completely Non-Trivial FD:** $X \to Y$ is completely non-trivial if $Y \cap X = \emptyset$.

Examples

$$R(A,B,C,D),\ \Sigma = \{AB \to D,\ C \to A\}$$

- $AB \to D$: AB functionally determines D.
- $C \to A$: C determines A.

Keys in Relations

- Superkey - A set of attributes $S \subseteq R$ is a **superkey** if $S \to R$.
- Candidate Key - A **candidate key** is a minimal superkey: $S \to R$ but no $S' \subset S \to R$.
- Prime Attribute - An attribute in any candidate key

Closures

Attribute Closure

The closure of a set S under Σ, denoted S^+, is the set of all attributes functionally dependent on S.

Algorithm for S^+:

1. Initialize $\Gamma = S$.
2. For $X \to Y \in \Sigma$, if $X \subseteq \Gamma$, add Y to Γ.
3. Repeat until no more attributes can be added.

Example:

$$R(A,B,C),\ \Sigma = \{A \to B,\ B \to C\}$$
$$\text{Compute } \{A\}^+ :\ \{A\} \to \{A,B,C\}.$$

Σ^+ Closures

The closure Σ^+ is the set of all FDs entailed by Σ.

Armstrong's Axioms

Rules

- **Reflexivity:** $Y \subseteq X \implies X \to Y$.
- **Augmentation:** $X \to Y \implies XZ \to YZ$ for $Z \subseteq R$.
- **Transitivity:** $X \to Y \wedge Y \to Z \implies X \to Z$.

Derived Rules

- **Union:** $X \to Y \wedge X \to Z \implies X \to YZ$.
- **Decomposition:** $X \to YZ \implies X \to Y \wedge X \to Z$.

Canonical Covers

Minimal Cover

A minimal cover Σ_m satisfies:

- Each FD is of the form $X \to A$ (single attribute on the right).
- The left-hand side of each FD is minimal.
- Removing any FD from Σ_m invalidates the cover.

Algorithm for Minimal Cover:

1. Decompose $X \to Y$ into $X \to A$ for all $A \in Y$.
2. Minimize X for each FD $X \to A$.
3. Remove redundant FDs.

Worked Example

Given:

$$R(A,B,C,D),\ \Sigma = \{AB \to C,\ C \to D,\ B \to D\}$$

Tasks:

- Find candidate keys:

$$AB^+ = \{A,B,C,D\} \implies AB \text{ is a candidate key.}$$

- Compute minimal cover:

$$\Sigma_m = \{AB \to C,\ C \to D,\ B \to D\}.$$

Equivalence of FD Sets

Two FD sets Σ_1 and Σ_2 are equivalent if:

$$\Sigma_1^+ = \Sigma_2^+.$$

Example:

$$\Sigma_1 = \{A \to B,\ B \to C\},\ \Sigma_2 = \{A \to C,\ A \to B\}$$
$$\Sigma_1^+ = \Sigma_2^+ \implies \Sigma_1 = \Sigma_2$$

Anomalies and Boyce-Codd Normal Form (BCNF)

Anomalies in Databases

Definitions

Anomalies occur when a schema violates functional dependencies (FDs), leading to redundancy and inconsistency.

Types of Anomalies

- **Redundancy:** Repeated data (e.g., faculty stored multiple times for each student).
- **Update Anomaly:** Failure to update all instances leads to inconsistencies.
- **Deletion Anomaly:** Deleting a record removes important data (e.g., removing the last student deletes the department).
- **Insertion Anomaly:** Inability to insert data without providing unrelated fields (e.g., requiring a student to create a department).

Relational Concepts and Functional Dependencies

A relational schema R consists of attributes $\{A,B,C,D,\dots\}$ and a set of functional dependencies Σ.

- A **functional dependency** $X \to Y$ implies that for any two tuples $t_1, t_2 \in R$, if $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$.
- Example:

$$R = \{A,B,C,D\}, \quad \Sigma = \{A \to B, B \to C, AC \to D\}.$$

$A \to B$ indicates that B is uniquely determined by A.

Example Table (Single Schema):

$$R(A,B,C,D) \quad \Sigma = \{A \to B, B \to C, C \to D\}$$

A	B	C	D
a1	b1	c1	d1
a2	b2	c2	d2

- **Redundancy:** $B \to C$ implies B values repeat unnecessarily.
- **Update Anomaly:** Changing $b1 \to c1$ requires updating all occurrences of $b1$.

Boyce-Codd Normal Form (BCNF)

A schema R is in BCNF if for every functional dependency $X \to Y$:

$$X \to Y \text{ is trivial (i.e., } Y \subseteq X) \text{ or } X \text{ is a superkey.}$$

Key Definitions:

- A **superkey** X uniquely identifies all attributes in R: $X \to R$.
- A **candidate key** is a minimal superkey.

Example:

$$R(A,B,C,D), \quad \Sigma = \{A \to B, B \to C, C \to D\}.$$

- Candidate key: A, since $A \to B, B \to C, C \to D$ implies $A \to R$.
- $B \to C$ violates BCNF since B is not a superkey.

BCNF Decomposition

To achieve BCNF:

1. Identify a dependency $X \to Y$ violating BCNF (i.e., X is not a superkey).
2. Decompose R into:

$$R_1 = X^+, \quad R_2 = (R - X^+) \cup X.$$

3. Repeat for R_1 and R_2 until all resulting tables satisfy BCNF.

Example:

$$R(A,B,C,D), \quad \Sigma = \{A \to B, B \to C, C \to D\}.$$

1. $B \to C$ violates BCNF.
2. Decompose:

$$R_1(B,C), \quad \Sigma_1 = \{B \to C\}, \quad R_2(A,B,D), \quad \Sigma_2 = \{A \to B, B \to D\}.$$

Properties of Decomposition

- **Lossless-Join:** The decomposition is lossless if $R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2$.
- **Dependency Preservation:** A decomposition preserves dependencies if $\Sigma = (\Sigma_1 \cup \Sigma_2)^+$.

Third Normal Form (3NF)

Motivation

- BCNF may lead to the loss of functional dependencies (**non-dependency preserving decomposition**).
- 3NF retains dependency preservation while minimizing redundancy.

Definition of Third Normal Form

A relation R with a set of functional dependencies Σ is in 3NF if for every functional dependency $X \to A$ in Σ^+:

$$X \to A \text{ is trivial (i.e., } A \subseteq X), \text{ or } X \text{ is a superkey, or } A \text{ is a prime attribute}$$

Key Terms:

- **Prime Attribute:** An attribute that is part of at least one candidate key.
- **Superkey:** A set of attributes X such that $X \to R$.
- **Candidate Key:** A minimal superkey.

Example:

$$R = \{A,B,C\}, \quad \Sigma = \{A \to B, B \to C\}.$$

- Candidate key: A (since $A \to B \to C$).
- $B \to C$: B is not a superkey, but C is a prime attribute.
- Conclusion: R is in 3NF but not in BCNF.

Properties of 3NF

- **Minimizes redundancy:** Prevents most anomalies while allowing certain controlled redundancies for dependency preservation.
- **Dependency preservation:** Ensures that all functional dependencies in Σ can be enforced in the decomposed relations.
- **Superset of BCNF:** Every BCNF relation is also in 3NF, but not all 3NF relations are in BCNF.

Algorithm: 3NF Synthesis (Bernstein Algorithm)

To decompose R into 3NF:

1. Compute a **minimal cover** Σ' of Σ:
   - Remove extraneous attributes from X in $X \to Y$.
   - Decompose non-minimal dependencies.
2. For each $X \to Y \in \Sigma'$, create a relation $R_i = X \cup Y$.
3. Ensure each candidate key of R is represented in at least one R_i.
4. Remove subsumed relations.

Example:

$$R = \{A,B,C,D\}, \quad \Sigma = \{A \to B, B \to C, AC \to D\}.$$

1. Minimal cover:

$$\Sigma' = \{A \to B, B \to C, AC \to D\}.$$

2. Relations:

$$R_1 = \{A,B\}, \quad R_2 = \{B,C\}, \quad R_3 = \{A,C,D\}.$$

3. Ensure candidate keys are preserved (e.g., $\{A,C\}$).

Comparison of 3NF and BCNF

- **3NF:** Allows redundancy to preserve all dependencies.
- **BCNF:** Removes redundancy but may sacrifice dependency preservation.

Hierarchy of Normal Forms:

$$\text{4NF} \subseteq \text{BCNF} \subseteq \text{3NF} \subseteq \text{2NF} \subseteq \text{1NF}.$$

Advantages of 3NF

- Retains all functional dependencies.
- Minimizes update anomalies compared to 2NF.
- Easier to implement in relational database systems.