CS2102 Finals Cheatsheet v1.0 (2021-04-28)
by Aadit Rahul Kamat, page 1 of 1

1 SQL

- Comments: – or /**/
- Built in data types: boolean, integer, float8, numeric, char, varchar, text, date, timestamp, array, JSON
- Comparison with null value is unknown and arithmetic with null is null
- Check constraints: e.g. check day in (1, 2, 3, 4, 5) or check ((day >= 8) and (day <= 15))
- Constraint names: constraint <constraint_name> <constraint_type>
- Foreign key constraint violations:
  - NO ACTION: reject delete/update if it violates constraint (default option)
  - RESTRICT: similar to NO ACTION except that constraint checking cannot be deferred
  - CASCADE: propogates delete/update to referencing tuples
  - SET DEFAULT: updates foreign keys of referencing tuples to some default value
  - SET NULL: updates foreign keys of referencing tuples to null value
- Transaction:
  - Consists of one or more update/retrieval operations
    begin;
    ...

    commit;
- Distinct keyword: Remove duplicate records
- Set operations: If Q1 and Q2 are union compatible relations then
  - Q1 union Q2 :
    $$Q1 \cup Q2$$
  - Q1 intersect Q2:
    $$Q1 \cap Q2$$
  - Q1 except Q2:
    $$Q1 - Q2$$
  - union, intersect and except remove duplicate records by themselves and preserve duplicate themselves when all keyword is appended e.g. union all
- Subquery expressions:
  - EXISTS subqueries: EXISTS (subquery)
  - IN subqueries: expression IN (subquery)
  - ANY/SOME subqueries: expression operator ANY (subquery)
  - ALL subqueries: expression operator ALL (subquery)
- Database modifications with subqueries: e.g.
  insert into Enrolls (sid, cid)
  select studentId, 101 from Students where year = 1
- Aggregate functions: For empty relation and relation with all null values, aggregate functions like min, max output null but count outputs 0 and n respectively
- GROUP BY clause properties: For each column A in relation R that appears in the SELECT clause, one of the following conditions must hold:
  - A appears in the GROUP BY clause,
  - A appears in an aggregated expression in the SELECT clause (e.g., min(A))
  - the primary key of R appears in the GROUP BY clause
  - if an aggregate function appears in the SELECT clause and there is no GROUP BY clause, then the SELECT clause must not contain any column that is not in an aggregated expression
- HAVING clause properties: For each column A in relation R that appears in the HAVING clause, one of the following conditions must hold:
  - A appears in the GROUP BY clause,
  - A appears in an aggregated expression in the HAVING clause, or
  - the primary (or a candidate) key of R appears in the GROUP BY clause
- Conceptual evaluation of Queries:
  select distinct select-list
  from from-list
  where where-condition
  group by groupby-list
  having having-condition
  order by orderby-list
  offset offset-specification
  limit limit-specification
  - Compute the cross-product of the tables in from-list
  - Select the tuples in the cross-product that evaluate to true for the where-condition
  - Partition the selected tuples into groups using the groupby-list
  - Select the groups that evaluate to true for the having-condition condition
  - For each selected group, generate an output tuple by selecting/computing the attributes/expressions that appear in the select-list
  - Remove any duplicate output tuples
  - Sort the output tuples based on the orderby-list
  - Remove the appropriate output tuples based on the offset-specification & limit-specification
- Common Table Expressions:
  with R1 as (Q1),
  R2 as (Q2),
  ...,
  Rn as (Qn)
  select/insert/update/delete statement S;
- Views: create view <view_name> as (<SQL statement>);
- Conditional Expressions
  - CASE expression:
    CASE
    WHEN condition_1 THEN result_1
    WHEN condition_2 THEN result_2
    WHEN ...
    ELSE else_result
    END
  - COALESCE
    * In built function that returns first non-null value in its arguments and null if all arguments are null
    * e.g. select name, coalesce(third, second, first) as result from Tests;
  - NULLIF
    * In built function that returns null if first argument is equal to second argument otherwise returns the first argument
    * e.g. select name, nullif(result, 'absent') as status from Tests;
- Pattern Matching with LIKE Operator
  - Underscore _ matches any single character
  - Percent % matches any sequence of 0 or more characters
  - "string **not like** pattern" is equivalent to "**not** (string **like** pattern)"
  - For more advanced regular expressions, use **similar to** operator

2 PSM

- General syntax:
  CREATE OR REPLACE FUNCTION/PROCEDURE
  <function_name> (<input_params with type>,
  <output_params with type>)
  RETURNS <function_type> AS $func$
  DECLARE
  ..
  BEGIN
  ..
  END
  $func$ LANGUAGE [sql|plpgsql];
  NOTE: use sql when you are executing purely SQL statements in the function body and plpgsql for other programmatic features such as assigning to variables and looping over records
- Cursor:
  - DECLARE curs CURSOR FOR (<sql statement>);
  - BEGIN:
    * open curs;
    * fetch [direction { from | in }] cursor_variable into target_variable;
    * close curs;

3 Triggers

- General syntax for Triggers:
  CREATE TRIGGER <trigger_name>
  AFTER|BEFORE|INSTEAD OF
  [INSERT|UPDATE|DELETE] ON <table_name>
  FOR EACH [ROW|STATEMENT] EXECUTE FUNCTION <trigger_function_name>
- General syntax for Trigger functions: CREATE OR REPLACE FUNCTION <trigger_function_name>
  (<input_parameters_with_types>,
  <output_parameters_with_types>)
  RETURNS TRIGGER AS $$
  DECLARE
  .. BEGIN
  .. END;
  $$
- Return values of Trigger Functions:
  - For a BEFORE INSERT trigger:
    * Returning a non-tuple t: t will be inserted
    * Returning a null tuple: no insertion operation will be performed
  - Similar for Update and Delete triggers
  - For After triggers, the return value does not matter
  - For an INSTEAD OF trigger:
    * Returning NULL: Ignore all operations on the current row
    * Returning non-NULL: Proceed as per normal
- Statement level triggers ignore the values returned by the trigger functions
- INSTEAD OF is only allowed on row-level while BEFORE/AFTER are allowed both on statement-level and row-level
- Trigger condition:
  CREATE TRIGGER
  ..
  WHEN (<condition>)
  ..
- Deferred Trigger:
  CREATE CONSTRAINT TRIGGER <trigger_name>
  ..
  DEFERRABLE [INITIALLY DEFERRED | INITIALLY IMMEDIATE]
  ..
- Order of Trigger Activation:
  - BEFORE statement -> BEFORE row -> AFTER row -> AFTER statement
  - Within each category, triggers are activated in alphabetic order
  - If BEFORE row returns NULL, subsequent triggers on the same row are omitted

4 Function Dependencies

- Armstrong's Axioms:
  1. Reflexivity: AB -> A
  2. Augmentation: If A ->B then AC -> BC
  3. Transitivity: If A -> B and B -> C then A -> C
- Algorithm for finding closure: Keep adding attributes that are activated directly or indirectly via FDs to the closure
- Algorithm for finding keys: Start from the smaller subsets and find those whose closures include all the attributes in the table. The attributes that do not appear on the right hand side of an FD have to be included in the key
- Prime attributes: Attributes that do not appear in a key

5 BCNF

- Decomposed FD: Has only one attribute on the right hand side
- BCNF Definition: A table R is in BCNF if and only if every non-trivial and decomposed FD has a superkey on its left hand side
- BCNF Check:
  1. Compute the closure of each subset
  2. Check for a "more but not all" closure i.e a closure that contains more attributes than the subset but not all the attributes.
  3. If one such closure does exist, the table is not in BCNF
- BCNF Decomposition:
  1. Find a subset X of attributes in R such that its closure contains more attributes than X but not all attributes
  2. Decompose R into two tables R1 and R2 such that R contains all attributes in X+ and R2 contains all attributes in X as well as the attributes not in X+
  3. Check if R1 and R2 are in BCNF, otherwise repeat Step 2 to decompose the tables further
  4. When deriving closures on decomposed tables, we project the original closure and remove attributes that do not appear in the decomposed table
- Lossless Decomposition: Common attributes in R1 or R2 that constitute a superkey of either tables

6 3NF

- Dependency Preservation: The FDs on the original table can be recovered from the FDs on the decomposed tables (derived from projected closure) i.e S', the set of FDs on decomposed tables is equivalent to S, the set of FDs on the original table.
- FD equivalence: Two sets of FDs S and S' are equivalent, if each FD in S can be derived from FDs in S' and vice versa.
- BCNF may not be dependency preserving.
- 3NF Definition: A table R is in 3NF if and only if every non-trivial and decomposed FD has a superkey on its left hand side:
  - Either the left hand side is a superkey
  - Right hand side is a prime attribute
- 3NF Check:
  1. Compute the closure of each subset
  2. Check for a "more but not all" closure i.e a closure that contains more attributes than the subset but not all the attributes such that at least one of the extra attributes is not a prime attribute
  3. If one such closure does exist, the table is not in 3NF
- 3NF Decomposition:
  1. Derive a minimal basis for the set of FDs on R
  2. In the minimal basis, combine the FDs whose left hand sides are the same
  3. Create a table for each FD
  4. If none of the tables contains a key for R, create a table that contains such a key. This allows lossless join of tables
  5. Algorithm for Minimal Basis:
     a) Transform the FDs into non-trivial and decomposed FDs
     b) Remove redundant attributes on the left hand side of each FD
     c) Remove redundant FDs