CS2102 Midterm Cheat-sheet

Part 1 Relational Algebra

Relational algebra operators:

Unary operators: selection σ, projection π, renaming ρ

Binary operators: cross-product $\times$, union $\cup$, intersect $\cap$, difference $-$

Part 2 Entity-relationship (ER) Diagram

1. Use square box for tables, use diamond box for relationship sets, use line to indicate relationship and multiplicity, use hollow circles to indicate attribute sets, use dark circles to indicate keys. An extra line should be draw for composite keys.
2. An extra line is needed for weak entities.
3. Cardinality is in the reverse direction to UML diagrams.
4. Three exceptions: one-to-many relationships – change the primary key of the joining table, one-to-one participation – remove the joining table, weak entity – use composite key.

Part 3 SQL Language

1. SQL built-in data types: boolean, integer, numeric (3, 1), real, char (50), varchar (60), text, date, timestamp
2. PostgreSQL specific data types: serial, with/without time-zone.
3. The domain of each data type includes a special value `NULL`.
4. SQL uses a three-valued logic system: true, false, unknown.
5. Two comparison predicate: `IS (NOT) NULL`, `IS (NOT) DISTINCT FROM`.
6. Data definition language (DDL):

```
CREATE TABLE Students(
studentId Integer unique REFERENCES Profiles(id),
name varchar(100) NOT NULL,
birthdate DATE,
age integer DEFAULT 20,
CHECK (age > 10)).
```

7. There can be composite primary keys/foreign keys/unique.
8. A foreign key constraint is: a subset of attributes `A` on referencing table refers to another subset of attributes `B` on referenced table. `B` must be declared as primary key or unique. `A` should contain one set of values of `B` or be `NULL`.
9. Data modifications:

```
INSERT INTO Students(name, id) VALUES('Bob', 67890);
DELETE FROM Students WHERE dept = 'Math';
UPDATE Accounts SET balance = balance * 1.2;
```

10. Schema modifications:

```
ALTER TABLE Students ADD COLUMN faculty varchar(20);
```

11. To achieve atomicity, each transaction begins with `BEGIN` and ends with `COMMIT/ROLLBACK`. All statements will be treated as one unit.
If one statement within the block fails, the whole transaction will be aborted. However, if we do not use the concept of transaction, let's say one of the statements fail, the else will still be committed.
12. Constraints are by default checked immediately (at the end of statement level), but you can optionally defer it to the end of transaction level.
13. To handle foreign key constraint violations:

- NO ACTION: rejects delete/update if it violates constraint (default option)
- RESTRICT: similar to *NO ACTION* except that constraint checking can't be deferred
- CASCADE: propagates 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

Notice that these types should be prepanded by `ON DELETE/UPDATE`.

14. join: inner join, natural join, left outer join, right outer join, full join.
15. Some useful functions: `SELECT 'price is ' || round(price/1.3) …`
16. `UNION, INTERSECT, EXCEPT` eliminate duplicate records, while `xxx ALL` preserves duplicate records.
17. `LIKE` operator: `WHERE name LIKE '___%e'`, underscore matches any single character, % matches any sequence of 0 or more characters.
18. Subquery expressions: `(NOT) EXISTS, IN, ANY/SOME, ALL, UNIQUE`.
19. Nested query/ sub-query.
20. Aggregate functions: `min, max, avg, sum, count, avg distinct, sum distinct, count distinct`.
21. Order-by & limit clause: in ascending order by default.

```
SELECT pizza, name FROM Sells ORDER BY price desc LIMIT 3;
```

22. View is a virtual relation: `CREATE VIEW xxx AS …;`
23. Group-by clause can be used wth having clause. However, the `SELECT / HAVING` clause for group by can includes those: 1) appear in the `GROUP BY` clause; 2) appear as an aggregate function; 3) `GROUP BY` includes a key.
24. To use aggregate functions without `GROUP BY` clause, then the select clause cannot contain anything except for aggregate functions.
25. Conceptual evaluation of queries:

1. Compute the cross-product of the tables in **from-list**
2. Select the tuples in the cross-product that evaluate to *true* for the **where-condition**
3. Partition the selected tuples into groups using the **groupby-list**
4. Select the groups that evaluate to *true* for the **having-condition** condition
5. For each selected group, generate an output tuple by selecting/computing the attributes/expressions that appear in the **select-list**
6. Remove any duplicate output tuples
7. Sort the output tuples based on the **orderby-list**
8. Remove the appropriate output tuples based on the **limit-specification**

26. Common table expression (CTE):

```
WITH XXX AS
…
SELECT …;
```

27. Conditional expressions: case

```
CASE ???
  WHEN … THEN …
  ELSE …
END
```

Part 4 Examples

1. Find the customer pairs (C1,C2) such that C1 < C2 and they like exactly the same pizzas

```
SELECT DISTINCT l1.cname AS cname1,
                l2.cname AS cname2
FROM Likes l1, Likes l2
WHERE l1.cname < l2.cname
AND NOT EXISTS (
   SELECT l3.pizza
   FROM Likes l3
   WHERE l3.cname = l1.cname
   AND NOT EXISTS (
      SELECT 1
      FROM Likes l4
      WHERE l4.cname = l2.cname
      AND l4.pizza = l3.pizza
   )
)
AND NOT EXISTS (
   SELECT l5.pizza
   FROM Likes l5
   WHERE l5.cname = l2.cname
   AND NOT EXISTS (
      SELECT 1
      FROM Likes l6
      WHERE l6.cname = l1.cname
      AND l6.pizza = l5.pizza
   )
)
```
