Data 101/Info 258: Data Engineering
Midterm Exam
Exam Reference Packet

UC Berkeley, Spring 2025

March 12, 2025

Name: ______________________________

Email: ______________________________@berkeley.edu

Student ID: ______________________________

Instructions

Do not open this exam reference packet until you are instructed to do so.

Make sure to write your name, email, and SID on this cover page.

1 PostgreSQL Reference

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
    [ HAVING condition ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ]
                          [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
```

where `from_item` can be one of:

```
    table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
                [ TABLESAMPLE sampling_method ( argument [, ...] ) ]
    [ LATERAL ] ( select ) [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    from_item join_type from_item { ON join_condition |
                            USING ( join_column [, ...] )
              [ AS join_using_alias ] }
    from_item NATURAL join_type from_item
    from_item CROSS JOIN from_item
```

and `grouping_element` can be one of: `expression` or `( expression [, ...] )`

and `with_query` is:

```
    with_query_name [ ( column_name [, ...] ) ] AS ( SELECT | VALUES )
```

1.1 Window Functions

```
<window or agg_func> OVER (
  [PARTITION BY <...>] [ORDER BY <...>] [RANGE BETWEEN range_start AND range_end] )
```

where `<window or agg_func>` can be one of:

```
    aggregate functions: AVG, SUM, etc., or:
    RANK() -- ordering within the window
    LEAD/LAG(exp, n) -- value of exp that is n ahead/behind in the window
    PERCENT_RANK() -- relative rank of current row as a %
    NTH_VALUE(exp, n) -- value of exp @ position n in window
```

and `range_start` and `range_end` can be one of:

```
    UNBOUNDED PRECEDING, UNBOUNDED FOLLOWING, CURRENT ROW,
    offset PRECEDING, offset FOLLOWING
```

1.2 Example Queries

```
SELECT id, location, age,
  AVG(age) OVER () AS avg_age
FROM residents;

SELECT id, location, age,
  SUM(age) OVER (
    PARTITION BY location
    ORDER BY age
    RANGE BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING ) AS a_sum
FROM residents
ORDER BY location, age;


CREATE TABLE <relation name> AS ( <subquery> );
CREATE TABLE zips (
    location VARCHAR(20) NOT NULL,
    zipcode INTEGER,
    in_district BOOLEAN DEFAULT False,
    PRIMARY KEY (location),
    UNIQUE (location, zipcode)
);

DROP TABLE [IF EXISTS] <relation name>;
ALTER TABLE zips
        ADD avg_pop REAL,
        DROP in_district;


CREATE TABLE cast_info (
  person_id INTEGER,
  movie_id INTEGER,
  FOREIGN KEY (person_id) REFERENCES actors (id)
    ON DELETE SET NULL ON UPDATE CASCADE,
  FOREIGN KEY (movie_id) REFERENCES movies(id) ON DELETE SET NULL
);
```

2 PostgreSQL String Utilities

String utility functions:

- `string || string` → text (concatenation)
- `SUBSTRING( string FROM start)` → text
- `SUBSTRING( string FROM re_pattern )` → text
- `SUBSTR( string, count )` → text
- `REPLACE(source, pattern, replacement)` → text
  In `REPLACE` pattern operates similar to `LIKE`, not a regular expression.
- `REGEXP_REPLACE(source, re_pattern, replacement, flags)` → text
  Note: `flags` must be `'g'` to execute a global match replacing all instances.
- SQL supports matching strings using two different types of pattern matching: SQL-style `LIKE` patterns, and POSIX Regular Expressions.
  - `string LIKE pattern` → boolean
  - `string ~ re_pattern` → boolean

Examples:

```
'Hello' || 'World' → 'HelloWorld'
STRPOS('Hello', 'el') → 2
SUBSTRING('Thomas' FROM 3) → 'omas'
SUBSTRING('Hello', 2, 3) → 'ell'
SUBSTR('Hello World', 7) → ‘World’
```

See the next page for Section 7: SQL Pattern Matching, which includes regular expressions.

3 SQL Pattern Matching

3.1 LIKE Patterns

SQL's `LIKE`, and `REPLACE` functions operate using a simplified pattern syntax.

```
'abc' LIKE 'abc' →   true                'abc' LIKE '_b_' →   true
'abc' LIKE 'a%'  →   true                'abc' LIKE 'c'   →   false
REPLACE('Hello World', 'l', 'L') → 'HeLLo WorLd'
```

If `pattern` does not contain percent signs (%) or underscores (_), then the pattern only represents the string itself; in that case `LIKE` acts like the equals operator. An underscore in pattern stands for (matches) any single character; a percent sign matches any sequence of zero or more characters.

3.2 Regular Expressions

This is an abbreviated reference which may prove helpful. The functions ~, `REGEXP_REPLACE`, and `SUBSTRING` accept `re_pattern` arguments which are regular expressions.

Escapes	Shorthand used in a match
`\d`	matches any digit
`\s`	matches any white space character
`\w`	matches any word character
Constraints	Used at the beginning or end of a match
`^`	matches at the beginning of the string
`$`	matches at the end of the string
Quantifier	Used after a match section
`*`	a sequence of 0 or more matches of the atom
`+`	a sequence of 1 or more matches of the atom
`?`	a sequence of 0 or 1 matches of the atom
`{m}`	a sequence of exactly m matches of the atom
`{m,}`	a sequence of m or more matches of the atom
`{m,n}`	a sequence of m through n (inclusive) matches of the atom; m cannot exceed n

```
'abcd' ~ 'a.c'      →  true         dot matches any character
'abcd' ~ 'a.*d'     →  true         * repeats the preceding pattern item
'abcd' ~ '(b|x)'    →  true         | means OR, parentheses group
'abcd' ~ '^a'       →  true         ^ anchors to start of string
'abcd' ~ '^(b|c)'   →  false
substring('foobar' from 'o.b')    → 'oob'
substring('foobar' from 'o(.)b')  →  'o'
substring('Thomas' from '...\$')  → 'mas'
regexp_replace('foobarbaz', 'b..', 'X')          → 'fooXbaz'
regexp_replace('foobarbaz', 'b..', 'X', 'g')     → 'fooXX'
regexp_replace('Hello World', '[aeiou]', '-', 'g') → 'H-ll- W-rld'
```