PostgreSQL

```
[ 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    ( expression [, ...] )

and with_query is:
    with_query_name [ ( column_name [, ...] ) ] AS ( select | values )
```

PostgreSQL, cont.

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

<window or agg_func>: aggregate functions: `AVG`, `SUM`, …, 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

range_start/range_end:

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

```
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;
```

```
REGEXP_REPLACE(source, pattern,
      replacement)
SELECT levenshtein(str1, str2) FROM Strings;
SELECT 'Hello' || 'World',
       STRPOS('Hello', 'el'),
       SUBSTRING('Hello', 2, 3);
```

```
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 Actor (id)
    ON DELETE SET NULL
    ON UPDATE CASCADE,
  FOREIGN KEY (movie_id)
    REFERENCES Movie (id)
    ON DELETE SET NULL);
```

Entity Resolution Diagrams (ER Diagrams)

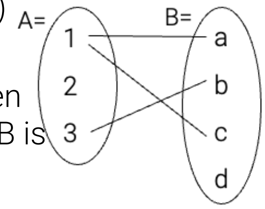
Entity set (rectangles)

- Entities: things, objects, etc.;
- Entity sets: sets entities w/commonalities.
- Every entity set is required to have a primary key (underlined attribute).

Attributes (ovals)
Atomic features connected to entity sets or relationships.

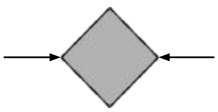
Relationships (diamonds)

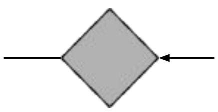
- Connects entity sets.
- A relationship between the entity sets A and B is a subset of A x B.

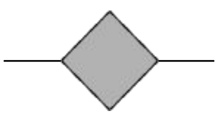


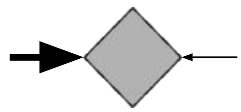
Edges in ER Diagrams can be directed/undirected and represent constraints on subset A x B.

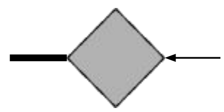
- Undirected edge (with no arrows): no constraints
- Directed edge (arrow): constrains, or determines, the relation to be at most one.
- Bolded edge determines the relation to be at least one.

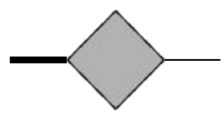

One-one: One on LHS connected to at most one of RHS, and vice-versa


Many-one: One on LHS connected to many on RHS


Many-many: One on LHS connected to many/few on the RHS, and vice versa


One-one: One on LHS connected to exactly one of RHS (≤ 1 & ≥ 1); one on RHS connected to at most one on LHS


Many-one: One on LHS connected to at least one on RHS; one on RHS connected to at most one on LHS


Many-many: One on LHS connected to at least one on RHS; RHS unconstrained

MongoDB

```
db.prizes.find({category: "peace"},
     {_id: 0, category: 1, year: 1,
        laureates.firstname: 1,
        laureates.surname: 1})
      .sort({year: 1, category: -1})
      .limit(2))
collection.find({})
collection.findOne({})

collection.aggregate ( [
  { stage: {…} },
  { stage: {…} },
  …
  { stage: {…} }
] )
```

where ***stage*** is one of

```
  $match
  $project
  $sort/$limit
  $group, e.g., { $group: [_id, "$item",
     totalqty, $sum: "$instock.qty" ] }
  $unwind, e.g., { $unwind: "$instock" }
  $lookup, e.g., { $lookup :
     {from : "inventory",
      localField : "instock.loc",
      foreignField : "instock.loc",
      as :"otheritems"}
     }
```

Odds and Ends

For a dataset X with median $\tilde{X} = \mathrm{median}(X)$, the Median Absolute Deviation (MAD) is $\mathrm{MAD}(X) = \mathrm{median}(|X_i - \tilde{X}|)$.

The Minimum Description Length (MDL) for encoding a set of values c in a set of types H is $\mathrm{MDL} = \min_{T \in H} \sum_{v \in c} (I_T(v) log(|T|) + (1 - I_T(v)) len(v))$

where $I_T(v)$ is an indicator for if v “fits” in type T (with |T| distinct values), log is base 2, and len(v) is the cost for encoding a value v in some default type.

A **functional dependency** (FD) is a form of constraint between 2 sets of attributes in a relation. For a relational instance with attributes X, Y, and Z:

- The FD X → Y is satisfied if for every pair of tuples t1 and t2 in the instance, if t1.X = t2.X, then t1.Y = t2.Y.
- The FD AB → C is satisfied if for every pair of tuples t1 and t2 in the instance, if t1.A = t2.A and t1.B = t2.B, then t1.C = t2.C.

`Map(k, v) → <k’, v’>*`

- Takes a key-value pair and outputs a set of key-value pairs
- There is one `Map` function call for each `(k,v)` pair

`Reduce(k’, <v’>*) → <k’, v’’>*`

- All values `v’` with same key `k’` are reduced together and processed in `v’` order
- There is one `Reduce` function call for each unique key `k’`