Introduction to relational data

A picture containing table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated with medium confidence

Table

Description automatically generated

Table

Description automatically generated

Graphical user interface, table

Description automatically generated

Table

Description automatically generated

**Psql Shell Commands**

* \list -> list all the databases
* CREATE DATABASE database\_name -> Replace database\_name with the name of the database
* \c database\_name -> connect to a database, replace database\_name with the database name you want to connect to
* CREATE TABLE table\_name (column data\_type) ->
  + CREATE TABLE people (pk SERIAL PRIMARY KEY, name VARCHAR(256) NOT NULL, height\_cm INT, gender VARCHAR(256),date\_of\_birth DATE);
  + CREATE TABLE address (pk SERIAL PRIMARY KEY, house\_number INT, street\_name VARCHAR(256), city VARCHAR(256), country VARCHAR(256));
* \d -> show all the tables in the database
* \d table\_name -> describe the column and data type of a table, replace table\_name with the actual table name
* ALTER TABLE table\_name ADD column\_name INT, ADD FOREIGN KEY (column\_name) REFERENCES table\_name\_to\_reference(pk\_column\_name); -> Add a foreign key to a table, replace table\_name with the name of the table, column\_name with the column name to be created, table\_name\_to\_reference with the targeted table to reference and pk\_column\_name with the column name of the primary key of that table.
  + ALTER TABLE address ADD people\_pk INT, ADD FOREIGN KEY (people\_pk) REFERENCES people(pk);

Table

Description automatically generated

* CREATE TABLE people (pk SERIAL PRIMARY KEY, name VARCHAR(256) NOT NULL, height\_cm INT, gender VARCHAR(256),date\_of\_birth DATE);
* CREATE TABLE address (pk SERIAL PRIMARY KEY, house\_number INT, street\_name VARCHAR(256), city VARCHAR(256), country VARCHAR(256));
* ALTER TABLE address ADD people\_pk INT, ADD FOREIGN KEY (people\_pk) REFERENCES people(pk);
* ALTER TABLE people ADD address\_pk INT, ADD FOREIGN KEY (address\_pk) REFERENCES address(pk);
* CREATE TABLE cars (pk SERIAL PRIMARY KEY, engine\_size\_cc INT, colour VARCHAR(256), manufacturer VARCHAR(256), model VARCHAR(256), year INT, people\_pk INT, FOREIGN KEY (people\_pk) REFERENCES people(pk));
* CREATE TABLE pets (pk SERIAL PRIMARY KEY, species VARCHAR(256), coat VARCHAR(256), age INT, people\_pk INT, FOREIGN KEY (people\_pk) REFERENCES people(pk));
* INSERT INTO people (name,height\_cm, gender, date\_of\_birth) VALUES ('Ben',167,'Male','1978-03-14');
* INSERT INTO pets (species, coat, age, people\_pk) VALUES ('cat', 'black', 3, 1);

Relational modelling limitations

**CSV type to table**

Table

Description automatically generated

CREATE TABLE csv\_data (pk SERIAL PRIMARY KEY, id VARCHAR(100), source VARCHAR(5), type VARCHAR(20), start INT, stop INT);

**People and connection to table**

A close-up of a stethoscope

Description automatically generated with low confidence

By Splitting to smaller network for the above

Diagram

Description automatically generated with low confidence

CREATE TABLE small\_network (pk SERIAL PRIMARY KEY, name VARCHAR(100) NOT NULL, first\_connection\_pk INT, second\_connection\_pk INT);

CREATE TABLE bigger\_network (pk SERIAL PRIMARY KEY, name VARCHAR(100) NOT NULL, first\_connection\_pk INT, second\_connection\_pk INT, third\_connection\_pk INT);

**\* Note ^ structure will have to keep editing/create the table when there is new network, therefore, using a junction table can solve it**

CREATE TABLE people\_nodes (pk SERIAL PRIMARY KEY, name VARCHAR(100) NOT NULL);

CREATE TABLE connections (first\_pk INT, second\_pk INT);

**Family tree to table**

Diagram

Description automatically generated

CREATE TABLE family\_tree (pk SERIAL PRIMARY KEY, name VARCHAR(100) NOT NULL, parent\_one\_pk INT, parent\_two\_pk INT);

Database good practice

Normalisation is the process of correctly disaggregating data sets, such that each data point is atomic represented only a single time and can be efficiently represented by the relational model

Graphical user interface, text

Description automatically generated

Remove attributes column and make it to a table, make sequencing factory into a table

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

Table

Description automatically generated

CREATE TABLE ec (pk SERIAL PRIMARY KEY, EC\_name VARCHAR(256));

CREATE TABLE sequencing (pk SERIAL PRIMARY KEY, sequencing\_factory VARCHAR(256), factory\_location VARCHAR(256));

CREATE TABLE genes (pk SERIAL PRIMARY KEY, gene\_id VARCHAR(256) NOT NULL, entity VARCHAR(256), source VARCHAR(256), start INT, stop INT, sequencing\_pk INT, ec\_pk INT, FOREIGN KEY (sequencing\_pk) REFERENCES sequencing(pk), FOREIGN KEY (ec\_pk) REFERENCES ec(pk));

CREATE TABLE products (genes\_pk INT, type VARCHAR(256), product VARCHAR(256), FOREIGN KEY (genes\_pk) REFERENCES genes(pk));

CREATE TABLE attributes (pk SERIAL PRIMARY KEY, key VARCHAR(256), value VARCHAR(256));

CREATE TABLE gene\_attribute\_link (gene\_pk INT, attributes\_pk INT, FOREIGN KEY (gene\_pk) REFERENCES genes(pk), FOREIGN KEY (attributes\_pk) REFERENCES attributes(pk));

Simple inserts and simple queries

INSERT INTO sequencing(sequencing\_factory, factory\_location) VALUES('Sanger','UK');

INSERT INTO ec(ec\_name) values('oxidoreductase');

INSERT INTO genes(gene\_id, entity, source, start, stop, sequencing\_pk, ec\_pk) VALUES('Gene1', 'Chromosome', 'ena', 190, 255, 1, 1);

INSERT INTO ec(ec\_name) VALUES('transferase');

INSERT INTO genes(gene\_id, entity, source, start, stop, sequencing\_pk, ec\_pk) VALUES('Gene2', 'Chromosome', 'ena', 375, 566, CURRVAL('sequencing\_pk\_seq'), CURRVAL('ec\_pk\_seq'));

**\* Note CURRVAL is postgres keyword, return the latest value of a column**

INSERT INTO genes(gene\_id, entity, source, start, stop, sequencing\_pk, ec\_pk) VALUES('Gene3', 'Chromosome', 'ena', 780, 980, CURRVAL('sequencing\_pk\_seq'), 1), ('GENE4', 'Plasmid', 'ena' ,1001, 111, CURRVAL('sequencing\_pk\_seq'), CURRVAL('ec\_pk\_seq'));

INSERT INTO products (genes\_pk, type, product) VALUES(1,'gene','mrna'),(1,'cds','protein'),(2,'gene','mrna'),(2,'cds','protein'), (3,'gene','mrna'),(3,'cds','protein'), (4,'gene','mrna'),(4,'cds','protein');

INSERT INTO attributes(key,value) VALUES ('ID', 'genen:b001'), ('Name', 'thrL'), ('biotype', 'protein\_coding'), ('description' , 'thr operon leader peptide'),('Name', 'fucA'), ('description' , 'Fructokinase A'), ('Name', 'timB'), ('description', 'Triosphosphate isomerase'), ('Name', 'gluA'), ('description', 'Glucose isomerase');

INSERT INTO gene\_attribute\_link (gene\_pk, attributes\_pk) VALUES (1,1), (1,2), (1,3), (1,4), (2,3), (2,5), (2,6), (3,3), (3,7), (3,8), (4,3), (4,9), (4,10);

INSERT INTO genes(gene\_id, entity, source, start, stop, sequencing\_pk, ec\_pk) VALUES('Gene5', 'Plasmid', 'ena', 768, 888, 1, 2) RETURNING pk;

**\* Note RETURNING is a postgres keyword which will return the column value after the value is inserted**

Queries and table joins

SELECT \* FROM genes,ec; -> act as a normal join, will duplicate rows

SELECT \* FROM genes,ec WHERE genes.ec\_pk = ec.pk; -> act as a inner join

SELECT gene\_id,entity,start,stop,ec\_name,sequencing\_factory FROM genes, ec, sequencing WHERE genes.ec\_pk=ec.pk AND genes.sequencing\_pk=sequencing\_pk;

SELECT gene\_id, key, value FROM genes, attributes, gene\_attribute\_link WHERE genes.pk=gene\_attribute\_link.gene\_pk AND attributes.pk=gene\_attribute\_link.attributes\_pk;

A picture containing text, compact disk, screenshot

Description automatically generated

Query performance

**Creating index** -> CREATE INDEX index\_name on table(column\_name);

CREATE INDEX entity\_index on genes(entity);

SELECT \* FROM genes WHERE entity\_index = 'Chromosome';

**EXPLAIN AND ANALYZE**->

* EXPLAIN SELECT \* FROM genes,ec WHERE genes.ec\_pk=ec.pk; -> show the cost of the query
* EXPLAIN ANALYZE SELECT \* FROM genes,ec WHERE genes.ec\_pk=ec.pk; -> show the cost + execution time

**Optimize the join by doing subquery first** -> SELECT sub.\*, ec.\* FROM (SELECT \* FROM genes WHERE entity='Chromosome') sub JOIN ec ON sub.ec\_pk=ec.pk;

**OR**

**Denormalizing the table(combining table to reduce join)**-> CREATE TABLE genes\_with\_ec (pk SERIAL PRIMARY KEY, gene\_id VARCHAR(256) NOT NULL, entity VARCHAR(256), source VARCHAR(256), start INT, stop INT, sequencing\_pk INT, ec\_name VARCHAR(256), FOREIGN KEY (sequencing\_pk) REFERENCES sequencing(pk));

**OR**

**Create materialized view(create a table that take data from a table)**->

**CREATE MATERIALIZED VIEW materialized\_table\_name AS select\_statement;**

CREATE MATERIALIZED VIEW gene\_ec\_view AS SELECT gene\_id, entity, source, start, stop, sequencing\_pk, ec\_name FROM genes, ec WHERE genes.ec\_pk=ec.pk;

**\* Note this does not update the data if any changes is made on the original table, to update the data use this command**

**REFRESH MATERIALIZED VIEW materialized\_table\_name;**

REFRESH MATERIALIZED VIEW gene\_ec\_view;

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Altering the database

**DROP DATABASE database\_name;** -> Delete the database

**DROP TABLE table\_name;** ->Delete the table

**ALTER TABLE table\_name DROP COLUMN column\_name CASCADE;** -> Delete a column from a table, adding **CASCADE** will update the materialized view too

**ALTER TABLE table\_name ADD COLUMN column\_name data\_type;** -> Add a column into a table