SQL

Schwartz

November 8, 2017

Sizes of things

	Binary		Decimal		Example
Bit	2^0	1			Binary (0 or 1)
Byte (B)	2^3	8			"S" = 01010011
Kilobyte (KB)	2^{10}	1,024	10^3	1,000	Word Document
Megabyte (MB)	2^{20}	1,048,567	10^6	1,000,000	Digital Photo
Gigabyte (GB)	2^{30}	1,073,741,824	10^9	1,000,000,000	DVD
Terabyte (TB)	2^{40}	1,099,511,627,776	10^{12}	1,000,000,000,000	Hard Drive
Petabyte (PB)	2^{50}	1,125,899,906,842,624	10^{15}	1,000,000,000,000,000	Some of Facebook
All Atoms	2^{266}	$\cdots$	10^{80}	$\cdots$	Universe
TSP routes	2^{329}	$(71-1)!/2$	10^{99}	$\cdots$	71 cities

Sizes of things

	Binary		Decimal		Example
Bit	2^0	1			Binary (0 or 1)
Byte (B)	2^3	8			"S" = 01010011
Kilobyte (KB)	2^{10}	1,024	10^3	1,000	Word Document
Megabyte (MB)	2^{20}	1,048,567	10^6	1,000,000	Digital Photo
Gigabyte (GB)	2^{30}	1,073,741,824	10^9	1,000,000,000	DVD
Terabyte (TB)	2^{40}	1,099,511,627,776	10^{12}	1,000,000,000,000	Hard Drive
Petabyte (PB)	2^{50}	1,125,899,906,842,624	10^{15}	1,000,000,000,000,000	Some of Facebook
All Atoms	2^{266}	$\cdots$	10^{80}	$\cdots$	Universe
TSP routes	2^{329}	$(71-1)!/2$	10^{99}	$\cdots$	71 cities

Ascii is encoded as 8 bits

$$\sum_{i=0}^{7} b_i 2^i$$

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Objectives

1. Learn what a RDBMS is
2. Learn the ways tables can be joined
3. Learn some postgreSQL
   - create, alter, insert-delete-update, and drop tables
4. Learn more postgreSQL
   - SELECT
   - AS, DISTINCT
   - *, /, +, -, CONCAT, ROUND, CAST, COALESCE
   - WHERE, CASE WHEN THEN ELSE END
   - =, <, <=, >, >=, ! =, <>, AND, OR, BETWEEN, LIKE, IN
   - NULL, IS NULL, IS NOT NULL
   - FROM/JOIN ON, LEFT, RIGHT, FULL [OUTER]
   - GROUP BY, MAX, MIN, SUM, AVG, COUNT
   - HAVING, ORDER BY, LIMIT
   - (SELECT ...)
5. Practice, practice, practice...

Relational Database Management System (RDBMS)

- Efficient queries of data and relations therein

Relational Database Management System (RDBMS)

- Efficient queries of data and relations therein
- *Schema*: tables and typed data columns
  - *Keys*: data relationships

Relational Database Management System (RDBMS)

- Efficient queries of data and relations therein
- *Schema*: tables and typed data columns
  - *Keys*: data relationships
- *ACID*: reliability properties

A: Atomicity – "all or nothing"
C: Consistency – "remain in legal state"
I: Isolation – "appropriate independence"
D: Durability – "persistance" (non-volatile storage)

Relational Database Management System (RDBMS)

- Efficient queries of data and relations therein
- *Schema*: tables and typed data columns
  - *Keys*: data relationships
- *ACID*: reliability properties

  A: Atomicity – "all or nothing"
  C: Consistency – "remain in legal state"
  I: Isolation – "appropriate independence"
  D: Durability – "persistance" (non-volatile storage)

- psql \l \c <DB> \d [table]

Schema

```
CREATE DATABASE dbname;
CREATE TABLE users {
    id INTEGER PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER,
    city VARCHAR(255),
    state VARCHAR(2)
}
```

Schema

```
CREATE DATABASE dbname;
CREATE TABLE users {
    id INTEGER PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER,
    city VARCHAR(255),
    state VARCHAR(2)
}
```

- Whitespace doesn't matter

  (but it can help make code clearer)

Schema

```
CREATE DATABASE dbname;
CREATE TABLE users {
    id INTEGER PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER,
    city VARCHAR(255),
    state VARCHAR(2)
}
```

- Whitespace doesn't matter

  (but it can help make code clearer)
- Capitalization (often) doesn't matter

  (but it can help make code clearer)

Schema

```
CREATE DATABASE dbname;
CREATE TABLE users {
    id INTEGER PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER,
    city VARCHAR(255),
    state VARCHAR(2)
}
```

- Whitespace doesn't matter

  (but it can help make code clearer)
- Capitalization (often) doesn't matter

  (but it can help make code clearer)
- Don't look like a noob
  - follow ubiquitous conventions
  - write beautiful looking code

Schema *efficiency* (referenced "users" table on last slide!)

```
CREATE TABLE visits {
    id INTEGER PRIMARY KEY,
    created_at TIMESTAMP,
    user_id INTEGER REFERENCES users(id)
    -- place foreign keys on the "many"
    -- side of a one-to-many relationship
};
```

Schema *efficiency*

```
CREATE TABLE visits {
    id INTEGER PRIMARY KEY,
    created_at TIMESTAMP,
    user_id INTEGER REFERENCES users(id)
    -- place foreign keys on the "many"
    -- side of a one-to-many relationship
};
```

```
CREATE TABLE posts {
    id INTEGER PRIMARY KEY,
    title VARCHAR(255)
};
```

```
CREATE TABLE tags {
    id INTEGER PRIMARY KEY,
    tag VARCHAR(255)
};
```

```
CREATE TABLE posts_tags {
    post_id INTEGER REFERENCES posts(id),
    tag_id INTEGER REFERENCES tags(id)
    -- "Normalized" data only duplicates foreign keys
};
```

Schema *efficiency* example

Do you like *this?*

My *new* Jawdins	#fab
My *new* Jawdins	#shoes
My *new* Jawdins	#fashion
Subway shoes	#shoes
Subway shoes	#envy

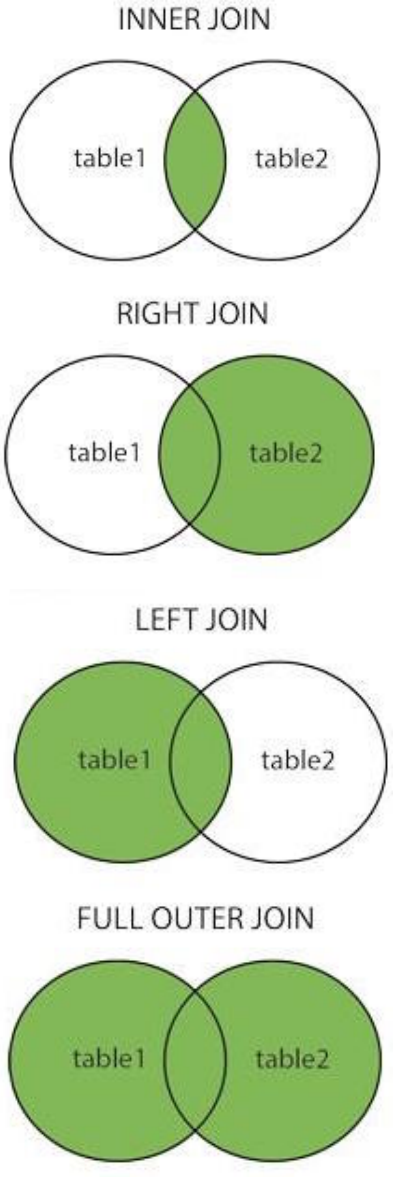
Or do you like *this?*

1	Tip-toein' in my Jawdins
2	NYC Subway Shoes

1	a
1	b
1	c
2	b
2	d

a	#fab
b	#shoes
c	#fashion
d	#envy

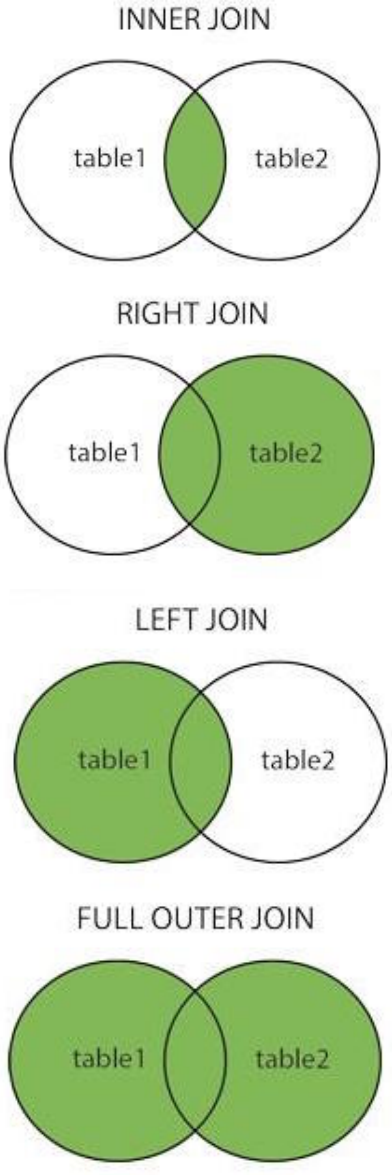
JOIN and *normalization* quiz



name	from	LL
ForesT	IL	T
JacK	VA	K
JamiE	CT	E
KatiE	CT	E
KeviN	NY	N
LaureN	NY	N
MichaeL	NY	L
RandalL	CA	L
RusselL	OH	L
TrevoR	NY	R

LL	place
N	CharlestoN
E	NashvillE
E	Portland, ME
R	Portland, OR
G	PittsburgH
E	Sante FE

JOIN and *normalization* quiz



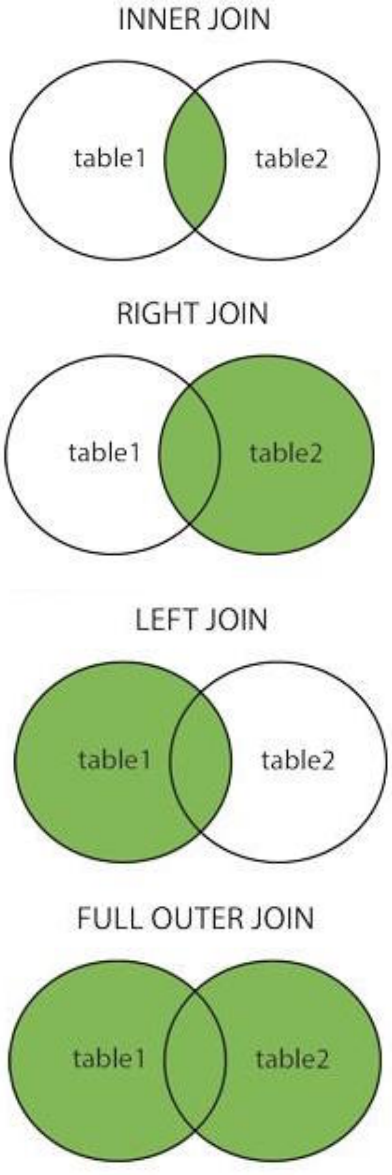
name	from	LL
ForesT	IL	T
JacK	VA	K
JamiE	CT	E
KatiE	CT	E
KeviN	NY	N
LaureN	NY	N
MichaeL	NY	L
RandalL	CA	L
RusselL	OH	L
TrevoR	NY	R

LL	place
N	CharlestoN
E	NashvillE
E	Portland, ME
R	Portland, OR
G	PittsburgH
E	Sante FE

Quiz: *how many rows result from*

- inner join?
- right join?
- left join?
- outer join?

JOIN and *normalization* quiz



name	from	LL
ForesT	IL	T
JacK	VA	K
JamiE	CT	E
KatiE	CT	E
KeviN	NY	N
LaureN	NY	N
MichaeL	NY	L
RandalL	CA	L
RusselL	OH	L
TrevoR	NY	R

LL	place
N	CharlestoN
E	NashvillE
E	Portland, ME
R	Portland, OR
G	PittsburgH
E	Sante FE

Quiz: *how many rows result from*

Quiz: *normalize the left table*

- inner join?
- right join?
- left join?
- outer join?

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- alter tables
- insert records
- update records
- delete records
- query records within and across tables

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- **create tables** (we saw this previously)
- alter tables
- insert records
- update records
- delete records
- query records within and across tables

```
CREATE DATABASE <dbname>;
CREATE [TEMPORARY] TABLE table AS <SQL query>;
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- **alter tables**
- insert records
- update records
- delete records
- query records within and across tables

```
ALTER TABLE table [DROP/ADD/ALTER] column [datatype];
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- **alter tables**
- insert records
- update records
- delete records
- query records within and across tables

```
DROP TABLE table;
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- alter tables
- **insert records**
- update records
- delete records
- query records within and across tables

```
INSERT INTO table [(c1,c2,c3,...)] VALUES (v1,v2,v3,...);
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- alter tables
- insert records
- **update records**
- delete records
- query records within and across tables

```
UPDATE table SET c1=v1,c2=v2,...WHERE cX=vX;
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- alter tables
- insert records
- update records
- **delete records**
- query records within and across tables

```
DELETE FROM table WHERE cX=vX;
```

Structured Query Language (SQL)

SQL is used to interact with RDBMS, allowing one to

- create tables (we saw this previously)
- alter tables
- insert records
- update records
- delete records
- **query records within and across tables**

`SELECT.FROM.JOIN.ON.WHERE.GROUP BY.HAVING.ORDER BY.LIMIT`

SQL *order of operations*

SQL *order of operations*

1. FROM/JOIN/ON
1. Merge Tables

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE

1. Merge Tables
2. Filter Rows

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY

1. Merge Tables
2. Filter Rows
3. Partition Rows

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”
5. HAVING

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows
5. Filter Aggregations

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”
5. HAVING
6. SELECT

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows
5. Filter Aggregations
6. Collect Columns

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”
5. HAVING
6. SELECT
7. “transform”

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows
5. Filter Aggregations
6. Collect Columns
7. Transform Columns

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”
5. HAVING
6. SELECT
7. “transform”
8. ORDER BY

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows
5. Filter Aggregations
6. Collect Columns
7. Transform Columns
8. Sort Rows

SQL *order of operations*

1. FROM/JOIN/ON
2. WHERE
3. GROUP BY
4. “aggregate”
5. HAVING
6. SELECT
7. “transform”
8. ORDER BY
9. LIMIT

1. Merge Tables
2. Filter Rows
3. Partition Rows
4. Aggregate Rows
5. Filter Aggregations
6. Collect Columns
7. Transform Columns
8. Sort Rows
9. Print Subset

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How do all queries start?

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: And then?

SELECT

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: And then? And then?

SELECT *

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: And then? And then? And then?

SELECT *

FROM

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What about the "*", again?

```
SELECT *

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What about the "*", again?

SELECT c1,c2,

FROM table

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What about the "*", again?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%')

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN –

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN –THEN 'a'
    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b'
    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What is the CASE syntax?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END
    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's the difference between the aliases?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table AS t
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's the difference between the aliases?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 AS t2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Now what?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Now what?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How else?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON (table.id = table2.id2)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How else?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How else?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON (id = id2)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How else?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON id = id2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: How else?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70 AND
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70 OR
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70 OR t2.c4
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this? What's next?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70 OR t2.c4 LIKE
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE t.c4 <= 70 OR t2.c4 LIKE 'S%'
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this? What's next?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: This actually doesn't work. Why?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's the difference?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT OUTER JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Now What's the difference?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                AND CASE - - - END
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                AND CASE - - - END OR
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's "NULL"?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                AND CASE - - - END OR t2.id2 IS NULL
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Do we need this?

```
SELECT c1,c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Do we need this? And this?

```
SELECT    c2,CONCAT(ROUND(100*c3/CAST(c2 AS REAL),2),'%'),
   CASE WHEN -THEN 'a' WHEN - THEN 'b' ELSE 'c' END AS clm
   FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
      WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
            (AND CASE - - - END OR t2.id2 IS NULL)
      GROUP BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Do we need this? And this?

```
SELECT CONCAT(MIN(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT CONCAT(MAX(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(1) >
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(1) >
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Can we do this?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING AVE(c1) >
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What are we doing here?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING AVE(c1) > (SELECT DISTINCT COUNT(*) FROM t3
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: You will also see this sometimes

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING AVE(c1) > (SELECT DISTINCT COUNT(1) FROM t3
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's the difference here?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this? What's next?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What's this? What's next?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm

    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)

        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)

        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query:

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
    ORDER BY
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: Why this?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
    ORDER BY c2
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: This?

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
    ORDER BY 1
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What are we MISSING??

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
    ORDER BY 1
    LIMIT 1
```

Declarative Language: *say what – not how*

The details of how things are actually done is just left up to SQL

My never ending query: What are we MISSING??

```
SELECT CONCAT(AVG(ROUND(100*c3/CAST(c2 AS REAL),2)),'%'),
    CASE WHEN –THEN 'a' WHEN – THEN 'b' ELSE 'c' END AS clm
    FROM table t LEFT JOIN table2 t2 ON (t.id = t2.id2)
        WHERE (t.c4 <= 70 OR t2.c4 LIKE 'S%') AND clm IN ('a','c')
                (AND CASE - - - END OR t2.id2 IS NULL)
        GROUP BY c2
        HAVING AVE(c1) > (SELECT COUNT(DISTINCT c5) FROM t3
                                WHERE c5 BETWEEN 'J' AND 'M')
    ORDER BY 1
    LIMIT 1;
```

Conclusion (and SUPER HINT)

It doesn't cost anything to

CREATE TABLE table AS (SELECT ...)

use it, and then

DROP TABLE table

And there's also that TEMPORARY thing that's totes cool/usable