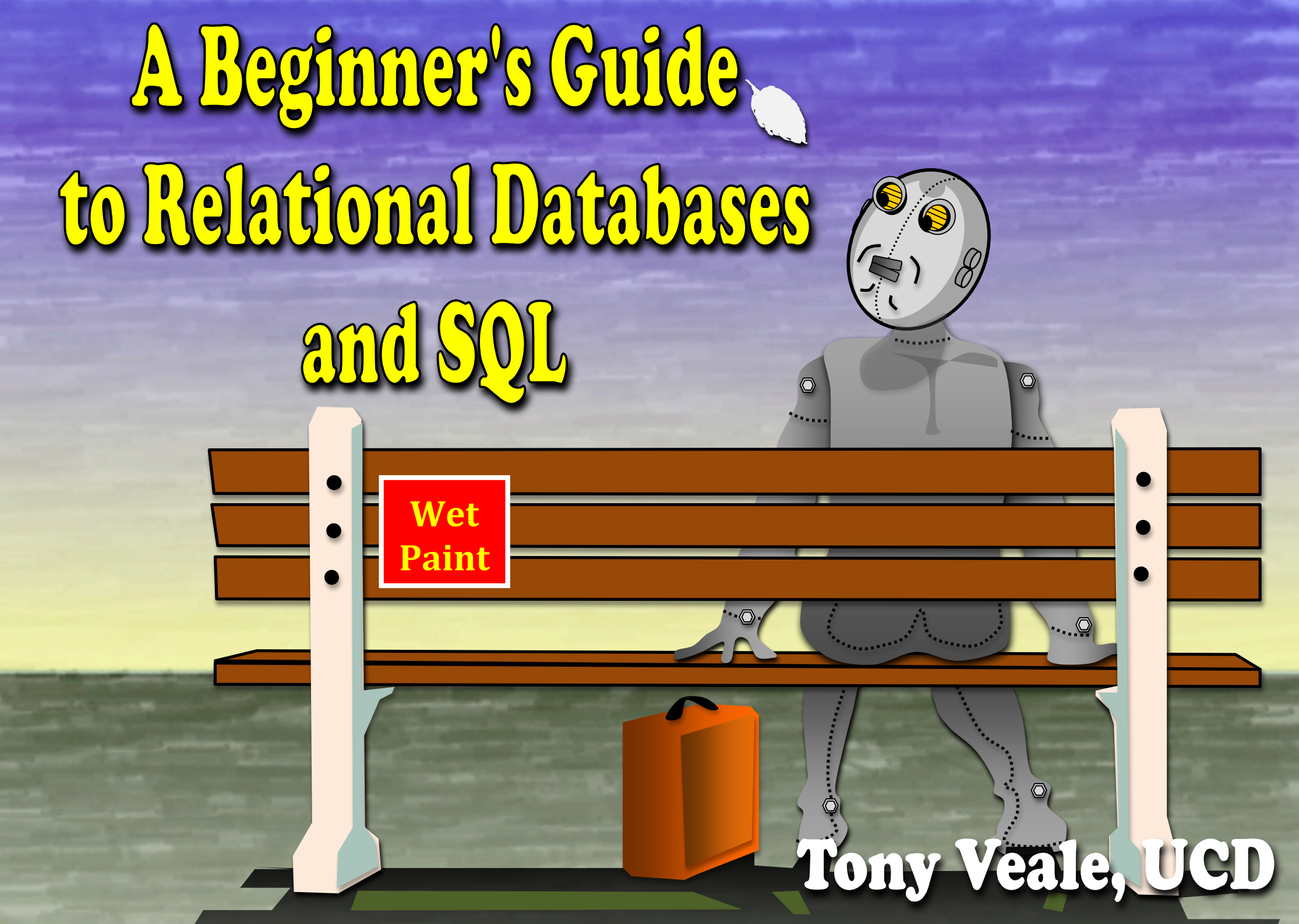
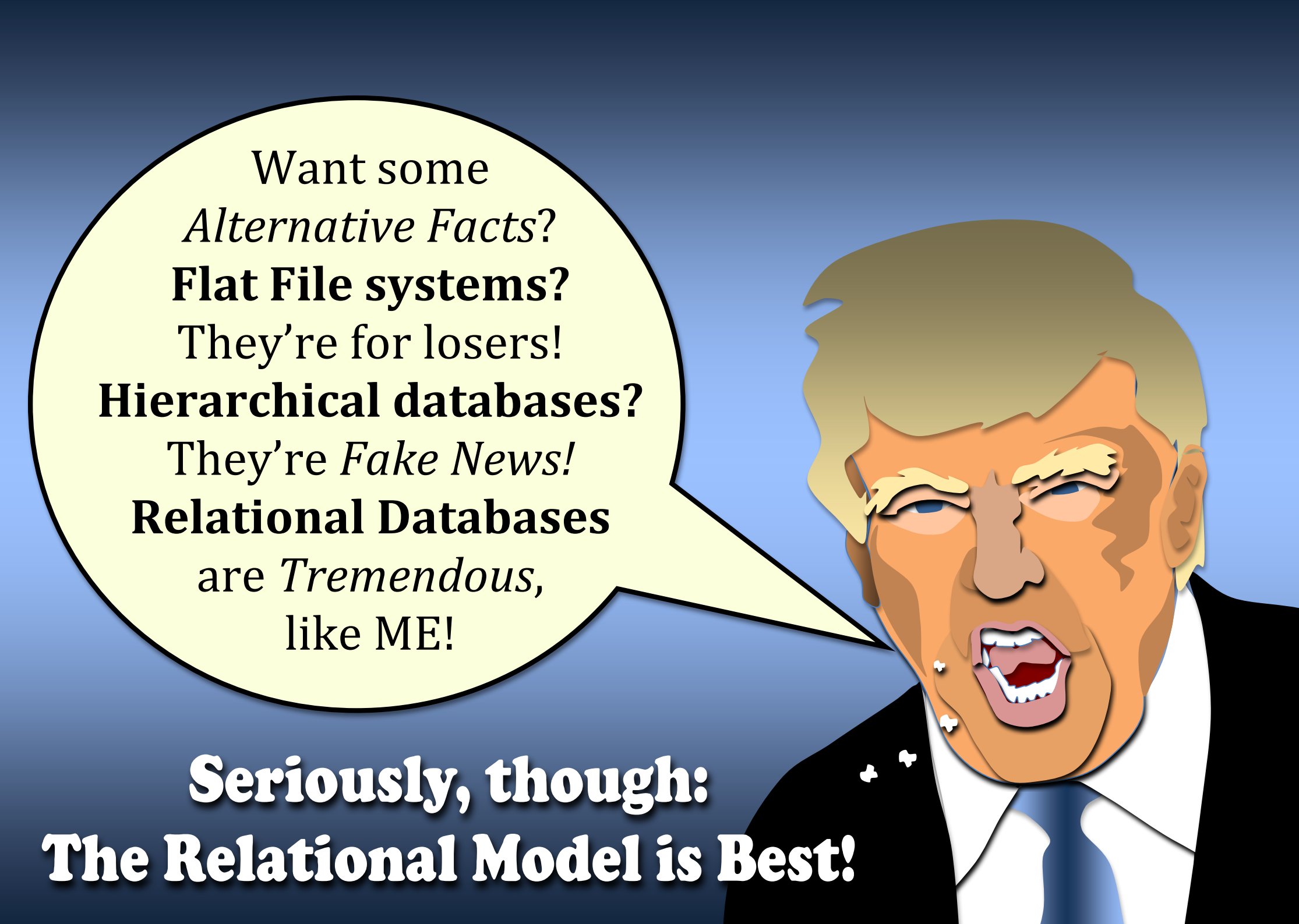
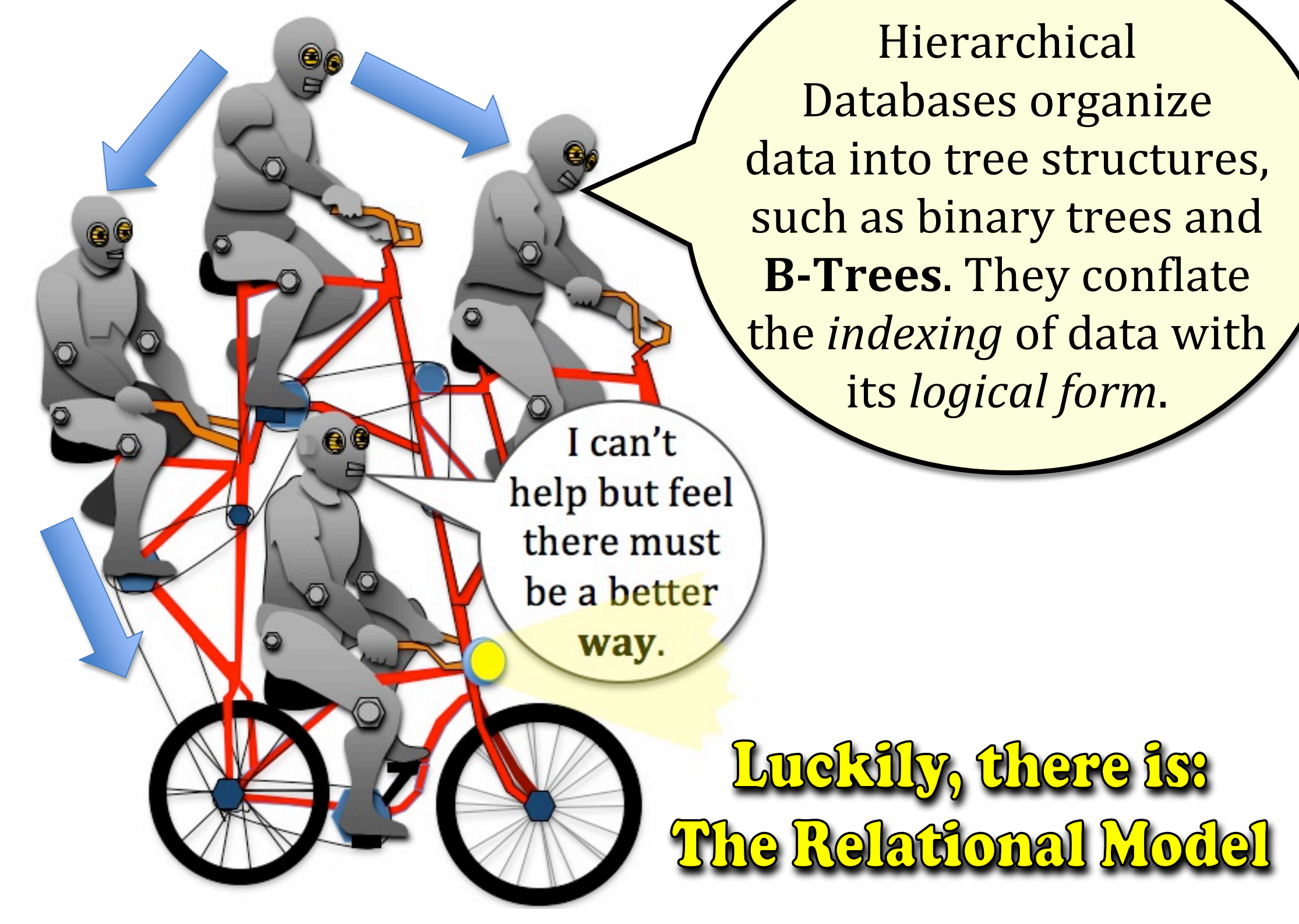
A Beginner's Guide to Relational Databases and SQL



Tony Veale, UCD


Want some
Alternative Facts?
Flat File systems?
They're for losers!
Hierarchical databases?
They're *Fake News!*
Relational Databases
are *Tremendous*,
like ME!
Seriously, though:
The Relational Model is Best!


Hierarchical Databases organize data into tree structures, such as binary trees and **B-Trees**. They conflate the *indexing* of data with its *logical form*.
I can't help but feel there must be a better way.
Luckily, there is:
The Relational Model

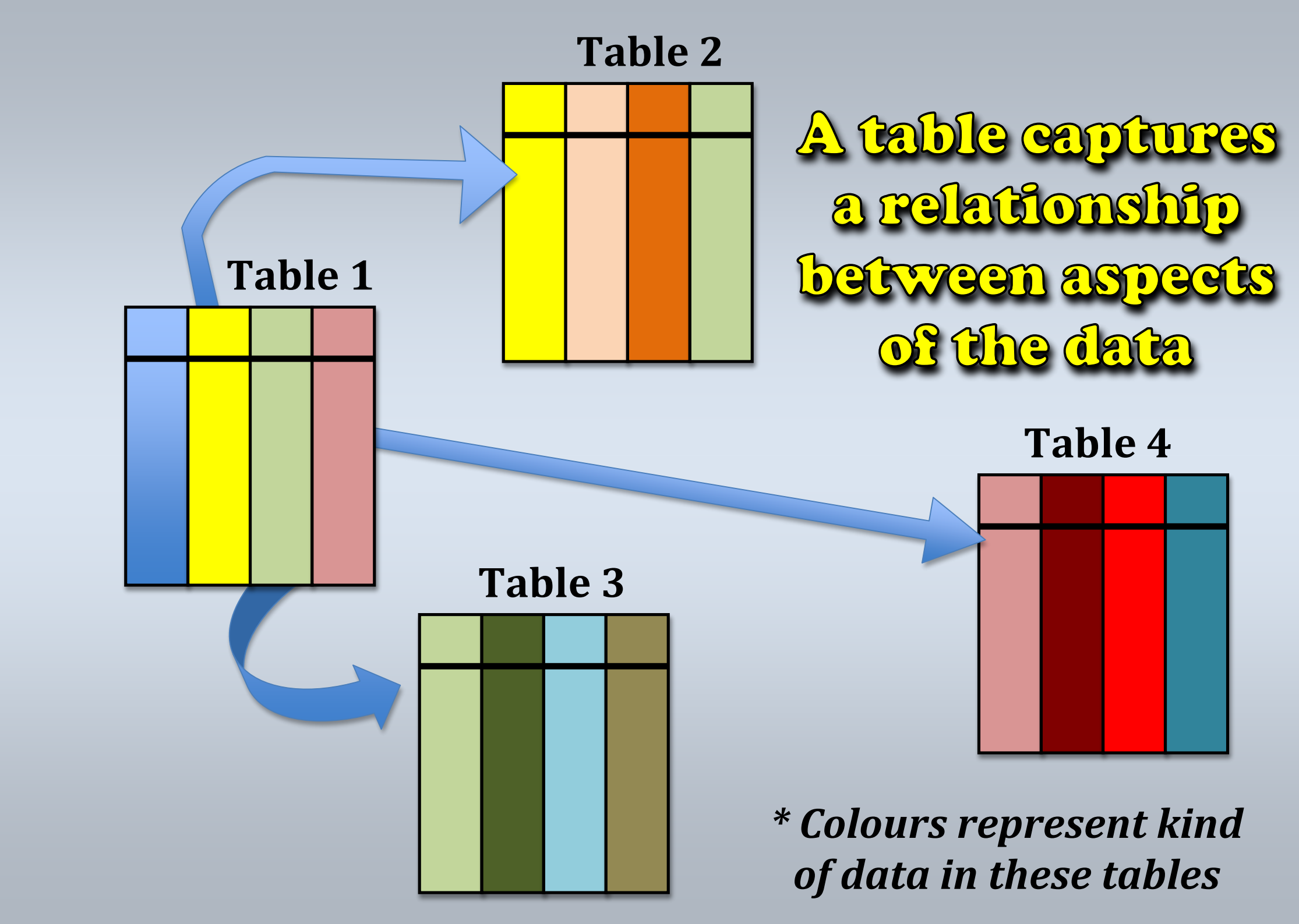


Table 2
Table 1
A table captures a relationship between aspects of the data
Table 4
Table 3
* Colours represent kind of data in these tables

The Relational Model captures the relationships in our data

This is a "Table"

Name	Opponent
Adam Smith	*Karl Marx*
Abraham Lincoln	*John Wilkes Booth*
Hillary Clinton	*Donald Trump*
Al Capone	*Eliot Ness*
Batman	*The Joker*

It consists of Columns and Rows

A Database that employs the Relational Model is called an RDBMS (Relational DB Mgmt. System)

Each table consists of columns calls "fields" such as Name, Gender, Fictive & Opponent

Name	Gender	Fictive	Opponent
Adam Smith	*male*	*no*	*Karl Marx*
Princess Leia	*female*	*yes*	*Darth Vader*
Hillary Clinton	*female*	*no*	*Donald Trump*
Al Capone	*male*	*no*	*Eliot Ness*
Batman	*male*	*yes*	*The Joker*

Each table consists of rows calls "records" There is one for each unique key value

Every table can be viewed as A collection of individual records

Princess Leia	*female*	*yes*	*Darth Vader*

Or every table can be seen as a set of fields

Opponent
Karl Marx
Darth Vader
Donald Trump
Eliot Ness
The Joker

The relational value comes from the intersection of both fields and records


It Does Not Compute!
SQL Constraints
and DATA INTEGRITY !!
It's a NULL value,
not a blank!

An RDBMS Manager can impose the following constraints on the columns/fields of a table:

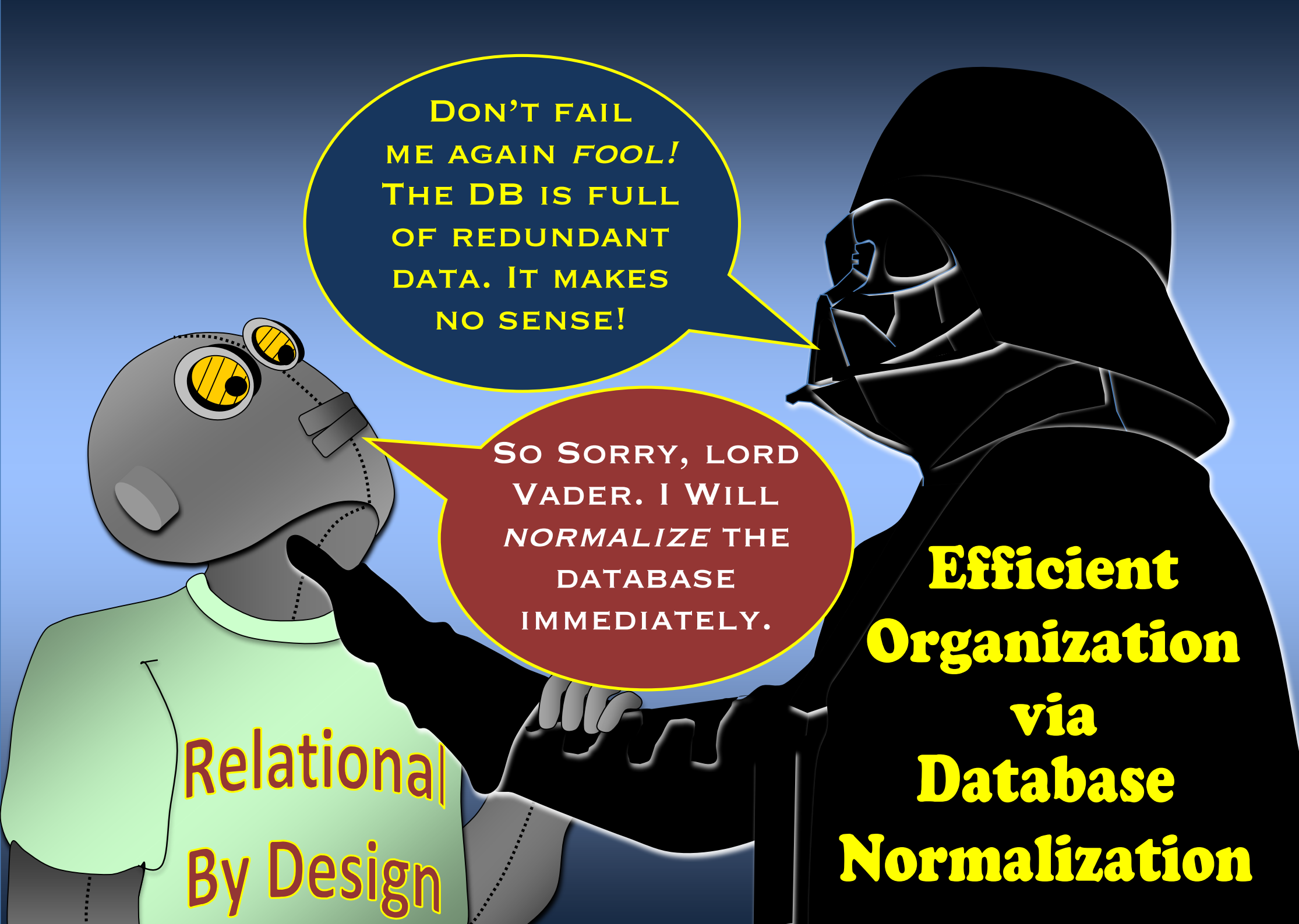
Constraint	Constraint Action
NOT NULL	*Ensures a field cannot ever have a NULL value*
DEFAULT	*Provides a default value for a field when none given*
UNIQUE	*Requires every value in a field/column to be different*
PRIMARY KEY	*Uniquely specifies every row/record in a table*
FOREIGN KEY	*Uniquely specifies every row/record in another table*
CHECK	*Provides an arbitrary test for the values of a field*
INDEX	*This field will be used to quickly retrieve data from table*

NB: These are "column"-level constraints
We can also apply constraints to whole tables

An RDBMS ensures "Data Integrity" by enforcing these constraints ...

Integrity Constraint	Constraint Action
ENTITY INTEGRITY	*There can be no duplicate records in a database table – EVER!*
DOMAIN INTEGRITY	*Every value of every field in every table must be of the right type, format and in the right range*
REFERENTIAL INTEGRITY	*A row/record cannot be deleted if it is being referenced by another row in another table*
USER-DEFINED INTEGRITY	*An RDBMS allows users to define own integrity constraints over tables and records*

Integrity is a normal state of affairs in an RDBMS because the database is "Normalized" as such


Don't fail me again *fool!* The DB is full of redundant data. It makes no sense!
So Sorry, lord Vader. I Will *normalize* the database immediately.
Relational By Design
Efficient Organization via Database Normalization

First Normal Form (**1NF**): *Ensure there are no redundant, repeating groups of data by imposing a unique* ***PRIMARY KEY*** *on every table.*

Second Normal Form (**2NF**):
2NF assumes INF, plus remove all **partial dependencies** of any field/column on the PRIMARY KEY.

Third Normal Form (**3NF**):
3NF assumes 2NF and 1NF. Ensures that every column/field in a table is dependent only on the PRIMARY KEY.

Normalization rasp ensures that data is stored logically and efficiently, to obviate future problems of ... *rasp* ... access

First Normal Form (1NF): Ensure no repeating groups of information, as in:

CUSTOMERS

Customer ID	Customer NAME	DOB	ADDRESS	ORDER
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine	*Light Sabre, model 27B/6*
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine	*Black helmet polish + cloth*
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine	*Lint remover for cape*

This table lacks a PRIMARY KEY: There is no way to uniquely specify each record

1NF: Split data into separate tables

(NB: Customer ID is a Foreign Key)

ORDERS

Order ID	Customer ID	Quantity	ORDER
ORD2001	THX1138	1	*Light Sabre, model 27B/6*
ORD2002	THX1138	1	*Black helmet polish + cloth*
ORD2003	THX1138	1	*Lint remover for cape*

CUSTOMERS

Customer ID	Customer NAME	DOB	ADDRESS
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine
PLO1001	Leia Organa	14.02.0045	23 Royal Lane, Aldebaran

Each table has its own PRIMARY KEY

2NF: No Partial Dependencies among fields

SALES

ORDER ID	Customer NAME	Customer ID	Order Details	Salesman
ORD2001	Darth Vader	**THX1138**	*Light Sabre, model 27B/6*	***Bib Fortuna***
ORD2171	Leia Organa	**PLO1001**	*Bunomatix Hair Curlers*	***Greedo Maxx***
ORD3092	Luke Skywalker	**LSW2003**	*Sand Speeder, model DSX74*	***Jabba Hutt***

Notice redundancy of Name and Details

These fields are redundant or dependent on others

Remove Customer Name (in CUSTOMERS table already)

SALES

ORDER ID	Customer NAME	Customer ID	Order Details	Salesman
ORD2001	Darth Vader	THX1138	*Light Sabre, model 27B/6*	***Bib Fortuna***
ORD2171	Leia Organa	PLO1001	*Bunomatix Hair Curlers*	***Greedo Maxx***
ORD3092	Luke Skywalker	LSW2003	*Sand Speeder, model DSX74*	***Jabba Hutt***

Remove Order Details (in ORDERS table already)

2NF removes redundancy, reduces inconsistency

3NF: All Non-primary fields are dependent on Key

CUSTOMERS

CUSTOMER ID	NAME	STREET	CITY	STATE	ZIP
THX1138	Darth Vader	17 Sith Plaza	Mos Eisley	Tatooine	90210
PLO1001	Leia Organa	23 Royal Lane	Windu City	Aldebaran	45872
OBI2007	Ben Kenobi	82 Rocky Way	Sandy Ave.	Tatooine	90734

No Transitive Dependencies between Non-Key fields

In the above table, all fields are dependent on the unique key. But aspects of the address are also dependent on each other. Since the Street or City or State and we must change ZIP too.

3NF: Remove Transitive Dependencies

ADDRESSES

ADDRESS ID	STREET	CITY	STATE	ZIP
9021017MET	17 Sith Plaza	Mos Eisley	Tatooine	90210
4587223WSA	23 Royal Lane	Windu City	Aldebaran	45872
9073482SAT	82 Rocky Way	Sandy Ave.	Tatooine	90734

CUSTOMERS

Customer ID	Customer NAME	DOB	ADDRESS ID
THX1138	Darth Vader	28.02.0024	**9021017MET**
PLO1001	Leia Organa	14.02.0045	**4587223WSA**

We link ZIP to Street & City & State in One Table

1NF: A Primary Key is required for every Table

2NF: Non-Key Attributes are dependent on whole key

3NF: Non-Key Attributes are dependent on nothing but the whole key*

* *The "whole key" refers to the possibility that a Primary Key may jointly comprise multiple fields in a row*

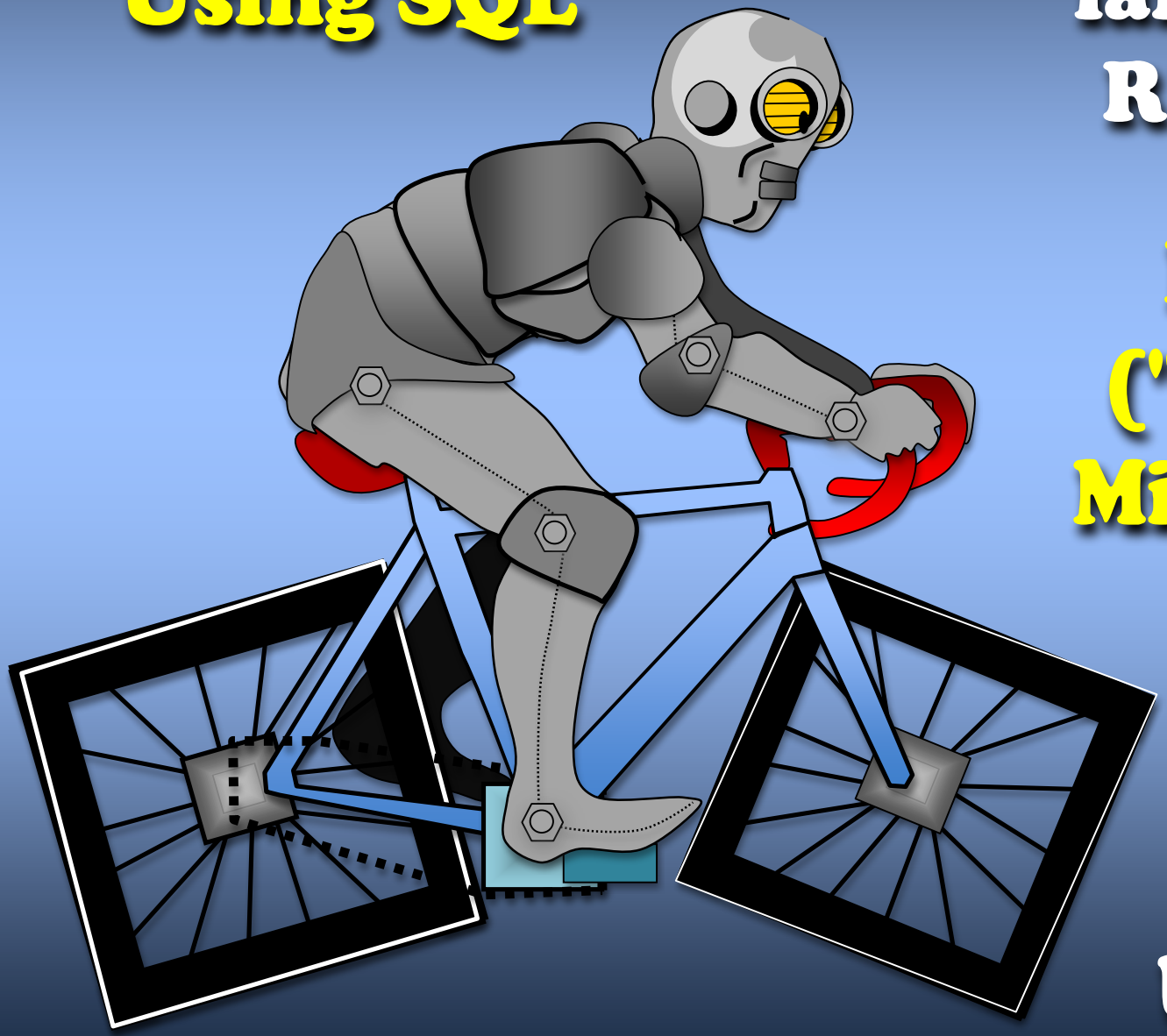
MySQL

A relational database employing the SQL structured query language

Free & Paid Versions

Efficient and Robust

Other RDBMS Solutions Using SQL



ORACLE
(founded 1977,
largest commercial
RDBMS provider)

MS SQL Server
("Sequel" Server)
Microsoft's RDBMS

Microsoft ACCESS
Entry-Level RDBMS
uses "Jet" SQL dialect

Data Definition Language (**DDL**):
Statements for creating a database and its tables; for deleting tables too

Data Manipulation Language (**DML**):
Statements for adding data to tables and for retrieving data from tables

Data Control Language (**DCL**):
Statements for granting and revoking privileges to users of the database

Transaction Control Language (**TCL**):
Statements for maintaining the state of the DB, via *commits* and *rollbacks*

SQL provides FOUR types of statements in four sub-languages ... rasp ... each of which has a different logical role in an RDBMS


It Lives!
@ @
DDL
Create A New Database and its Contents

Create a new SQL database:

```
CREATE DATABASE database_name;

USE database_name;
```

Drop (Remove) an existing SQL database:

```
DROP DATABASE database_name;
```

Create a New Table in a Database:

```
CREATE TABLE table_name(
    column1 datatype,
    column2 datatype,
    column3 datatype,
    .....
    columnN datatype,
PRIMARY KEY(one or more columns)
);
```

A table's PRIMARY KEY can combine multiple fields into a single unique key field

Create an index for your table for fast access:

```
CREATE UNIQUE INDEX index_name
    ON table_name ( column1,
                    column2,
                    ...,
                    columnN);
```

An index is a mapping from field values to the table rows that contain them

When an index is available, any SELECT query will use it for fast lookup of data

DROP a table and its contents from the database:

```
DROP TABLE table_name;
```

DROP an index for a table from the database:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

ADD a new column to a table:

```
ALTER TABLE table_name
ADD column_name datatype
```

DROP an existing column from a table:

```
ALTER TABLE table_name
DROP COLUMN column_name
```

RENAME a Table with a new name:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

Change the Data Type of a Table column:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype
```

INSERT a row of column values into a Table:

```
INSERT INTO table_name( col1, ...., colN)
VALUES ( value1, ..., valueN);
```

DML

Change the contents of a specific table row:

```
UPDATE table_name
SET col1 = value1, ..., colN=valueN
[ WHERE  {CONDITION} ];
```

DML

Delete any row matching Boolean criteria:

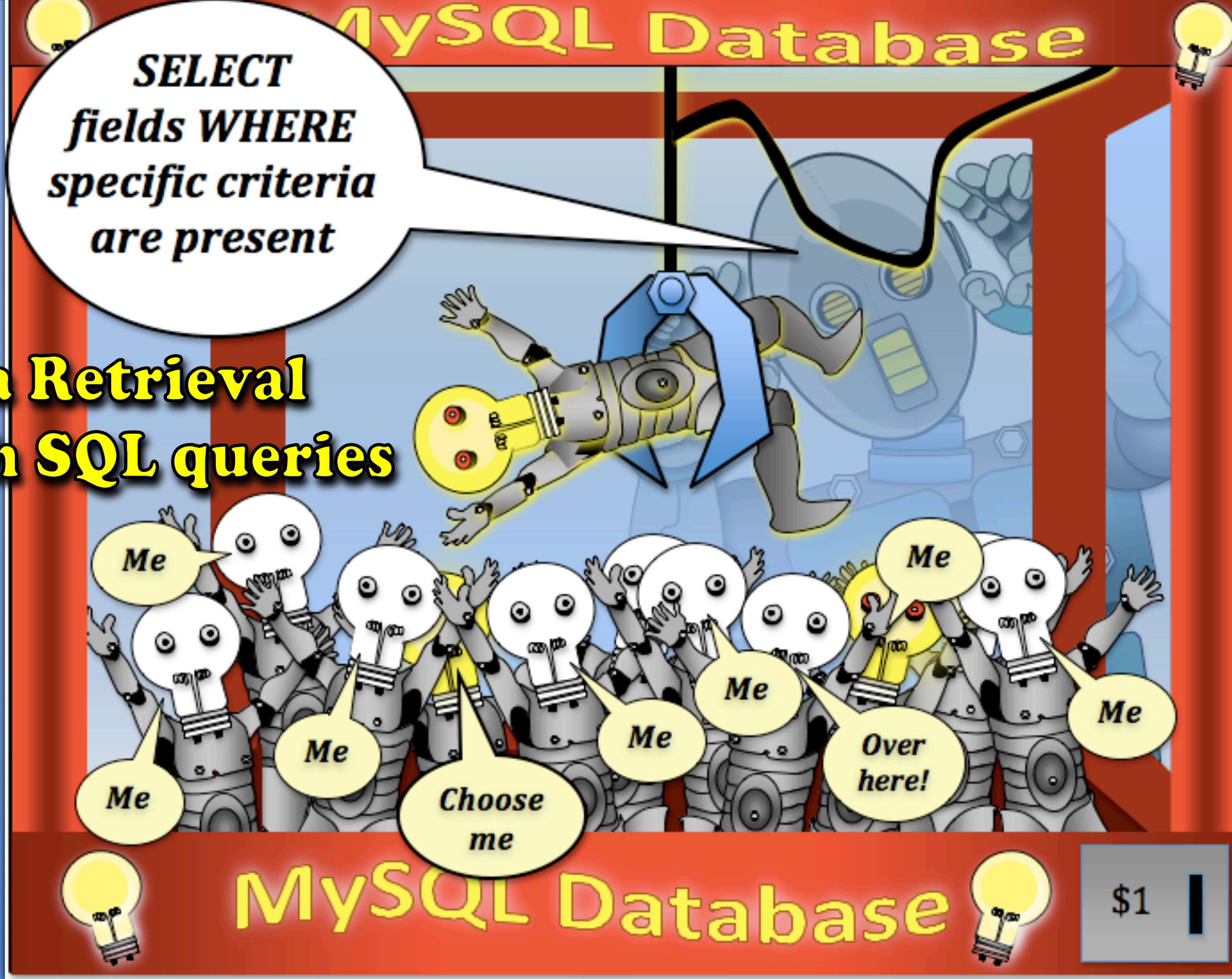
```
DELETE FROM table_name
WHERE {CONDITION};
```

DML

The WHERE clause specifies Boolean criteria:

```
[SELECT | DELETE FROM]
WHERE condition1
{and|or} condition2 ...
{and|or} ... conditionN;
```

DML

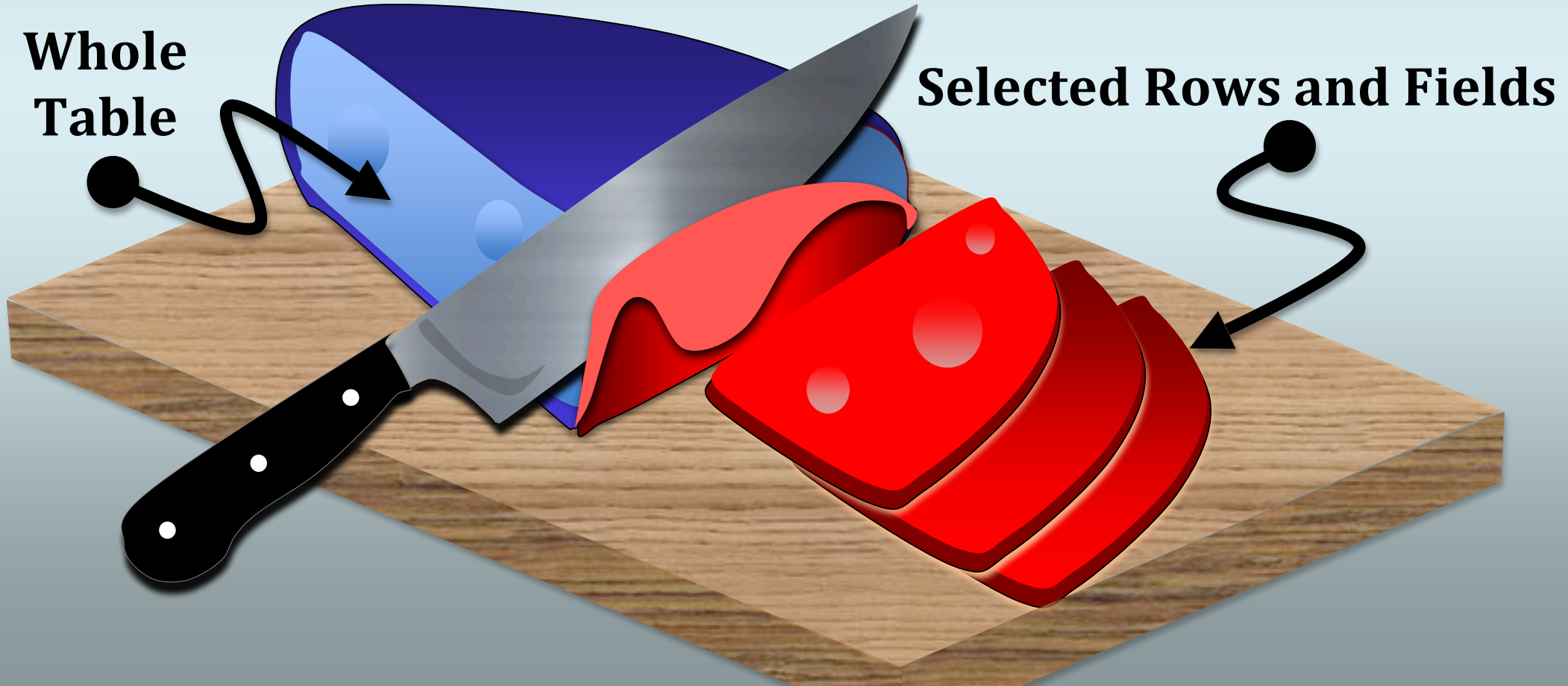


Data Retrieval with SQL queries


Using SQL for Queries
So what's your query, PIG?
SELECT Crime from NewsTable WHERE perp = you

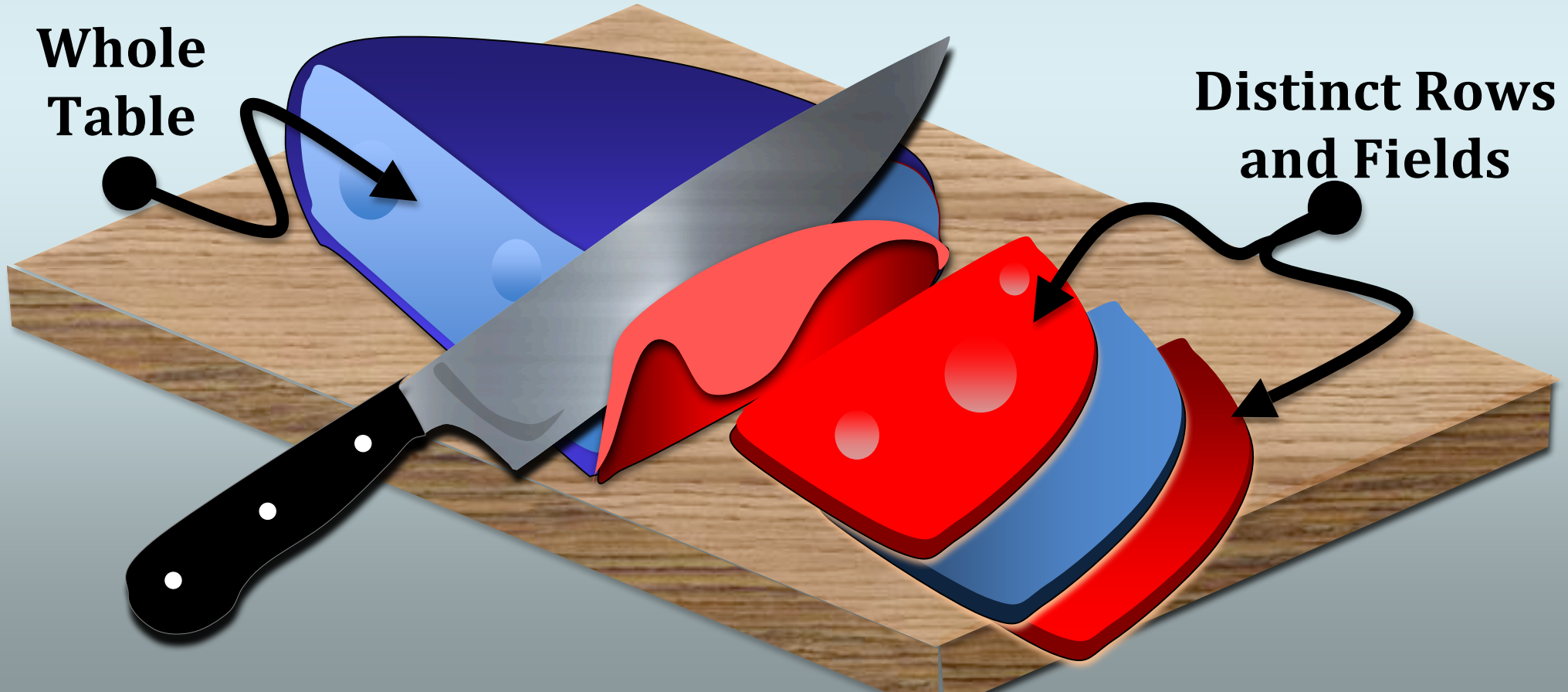
SELECT specific fields from a table:

```
SELECT column1, column2....columnN
FROM   table_name;
```



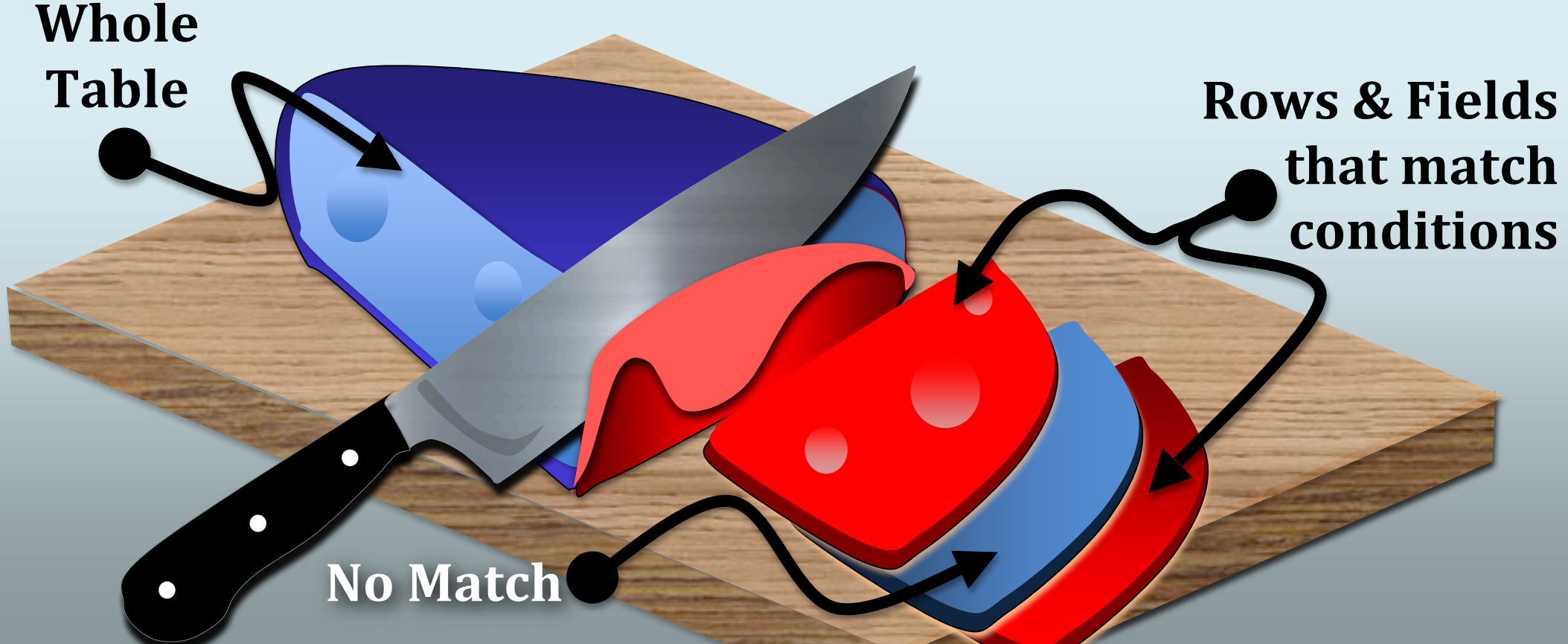
SELECT and filter for uniqueness of results:

```
SELECT DISTINCT column1....columnN
FROM   table_name;
```



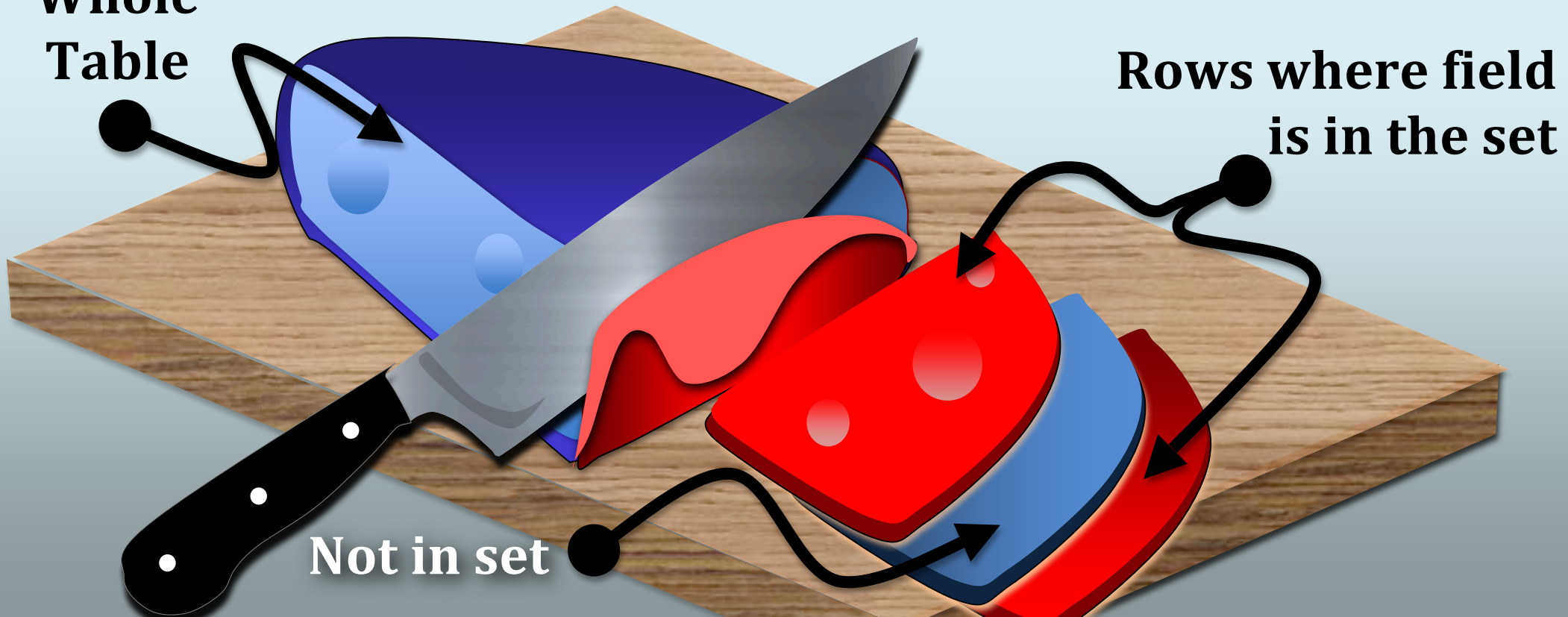
SELECT data that meets specific criteria:

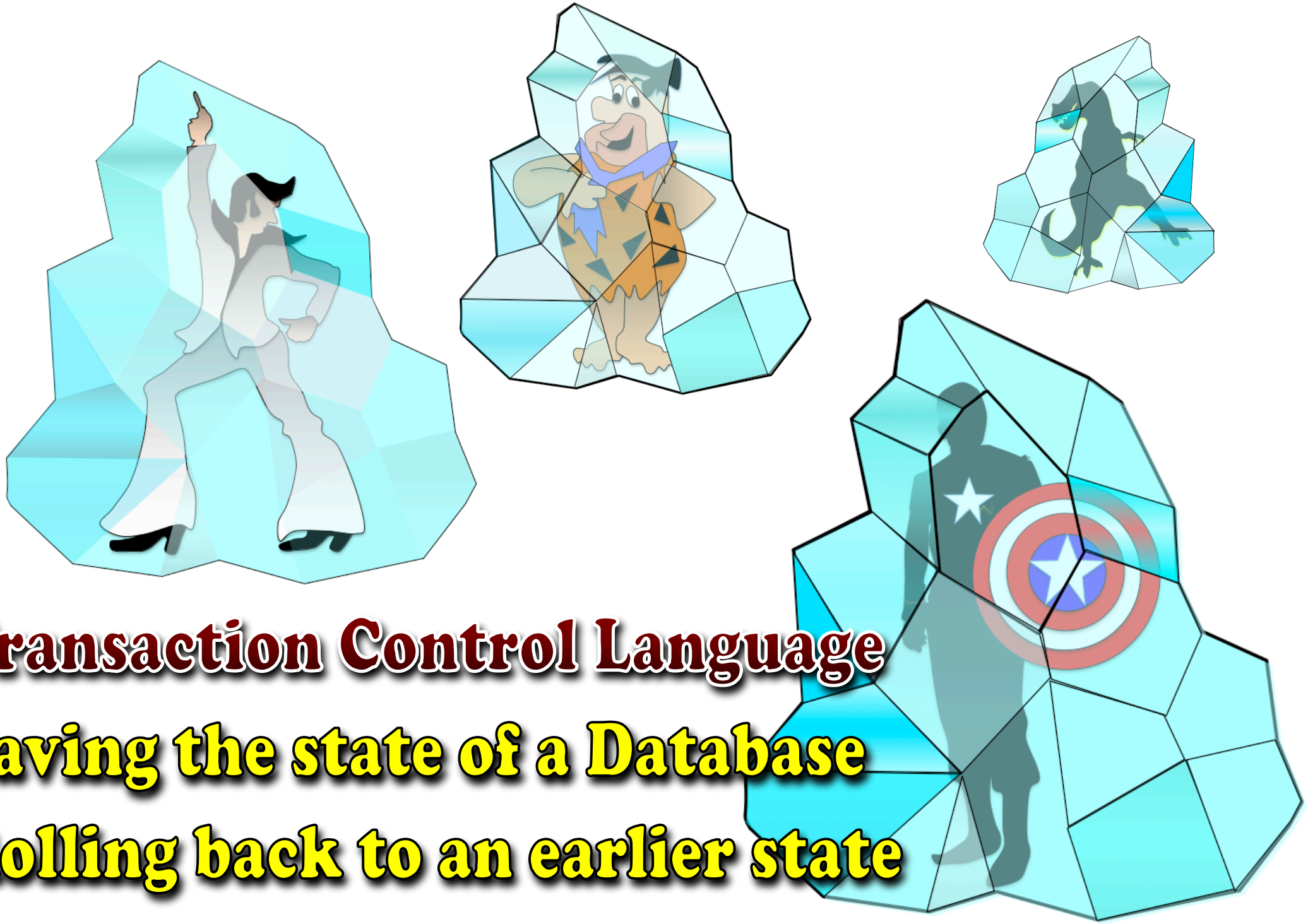
```
SELECT column1, ...., columnN
FROM   table_name
WHERE condition1 {and|or} conditionN
```



SELECT rows where field is in a set of values:

```
SELECT column1, column2....columnN
FROM   table_name
WHERE  column_name IN (val-1, ...,val-N);
```




Transaction Control Language
Saving the state of a Database
Rolling back to an earlier state

We can commit changes and mark the new state

```
COMMIT;                    (commit all changes to DB)
SAVEPOINT [StateName];
```

TCL

To undo changes, rollback back to an earlier state

```
ROLLBACK;                  (to last commit)
ROLLBACK to [StateName];
```

TCL

Exact Data Types for Columns/Fields:

DATA TYPE	FROM	TO
bigint	*-9,223,372,036,854,775,808*	*9,223,372,036,854,775,807*
int	*-2,147,483,648*	*2,147,483,647*
smallint	*-32,768*	*32,767*
tinyint	*0*	*255*
bit	*0*	*1*
decimal	*-10^38 +1*	*10^38 -1*
numeric	*-10^38 +1*	*10^38 -1*

Approximate Data Types for Fields:

DATA TYPE	FROM	TO
float	*-1.79E + 308*	*1.79E + 308*
real	*-3.40E + 38*	*3.40E + 38*

Date and Time Data Types for Fields:

DATA TYPE	FROM	TO
Date / datetime	*Jan 1, 1753*	*Dec 31, 9999*
smalldatetime	*Jan 1, 1900*	*Jun 6, 2079*
time	*00:00 AM*	*00:00 PM*

Characters and Strings Data Types:

DATA TYPE	DESCRIPTION
char	*Maximum length of 8,000 characters. (Fixed length non-Unicode characters)*
varchar	*Maximum of 8,000 characters. (Variable-length non-Unicode data).*
varchar(max)	*Maximum length of 231characters, Variable-length non-Unicode data (SQL Server).*
text	*Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.*

Unicode Characters / Strings Data Types:

DATA TYPE	DESCRIPTION
nchar	*Maximum length of 4,000 characters. (Fixed length Unicode)*
nvarchar	*Maximum length of 4,000 characters. (Variable length Unicode)*
nvarchar(max)	*Maximum length of 231characters (SQL Server only). (Variable length Unicode)*
ntext	*Maximum length of 1,073,741,823 characters. (Variable length Unicode)*

Binary Data Types in SQL:

DATA TYPE	DESCRIPTION
binary	*Maximum length of 8,000 bytes (Fixed-length binary data)*
varbinary	*Maximum length of 8,000 bytes. (Variable length binary data)*
varbinary(max)	*Maximum length of 231 bytes (SQL Server only). (Variable length Binary data)*
image	*Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)*

Miscellaneous Data Types in SQL:

DATA TYPE	DESCRIPTION
sql_variant	*Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.*
timestamp	*Stores a database-wide unique number that gets updated every time a row gets updated*
uniqueidentifier	*Store a globally unique identifier (GUID)*
xml	*Stores XML instances in a column or variable (SQL Server)*
cursor	*Reference to a cursor object.*
table	*Stores a result set for later processing*

Joining Multiple Tables Into One Virtual Table

A	B	C
a	b	c
d	e	f
g	h	i

×

X	Y	Z
o	p	q
r	s	t
u	v	w
x	y	z

=

Cartesian Join

A	B	C	X	Y	Z
a	b	c	o	p	q
a	b	c	r	s	t
a	b	c	u	v	w
a	b	c	x	y	z
d	e	f	o	p	q
d	e	f	r	s	t
d	e	f	u	v	w
d	e	f	x	y	z
g	h	i	o	p	q
g	h	i	r	s	t
g	h	i	u	v	w
g	h	i	x	y	z

If table #1 has C1 columns and R1 rows, and table #2 has C2 columns and R2 rows, then the joined table will have C1 + C2 columns and R1 × R2 rows.

Remember our Customer and Order Tables?

Order_ID	Customer_ID	Quantity	ORDER
ORD2001	THX1138	3	*Light Sabre, model 27B/6*
ORD2002	THX1138	1	*Black helmet polish + cloth*
ORD2003	PLO1001	2	*Bun-shaped hair curlers*

Customer_ID	Customer_NAME	DOB	ADDRESS
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine
PLO1001	Leia Organa	14.02.0045	23 Royal Lane, Aldebaran

We're going to Join them to access both at once

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers, Orders
WHERE Customers.Customer_ID = Orders.Customer_ID
AND Orders.Quantity > 1 ;
```

Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Leia Organa	Bun-shaped hair curlers	2

The Result Set Is A Projected Subset of the Join

Full Cartesian Join:

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers, Orders;
```

Notice the lack of join constraints above.

This query performs projection (column filtering) but not selection (row filtering)

The Result Set thus contains R1 x R2 rows where R1 = Number of Rows in Table 1 and R2 =Number of Rows in Table 2

A Database of Poker Hands and Playing Cards

Poker_Hands

Player_ID	Game_ID	R1	R2	R3	R4	R5
12789	17MET	QH	QS	3D	3C	3H
90734	82SAT	7C	4S	4D	4C	3H

Playing_Cards

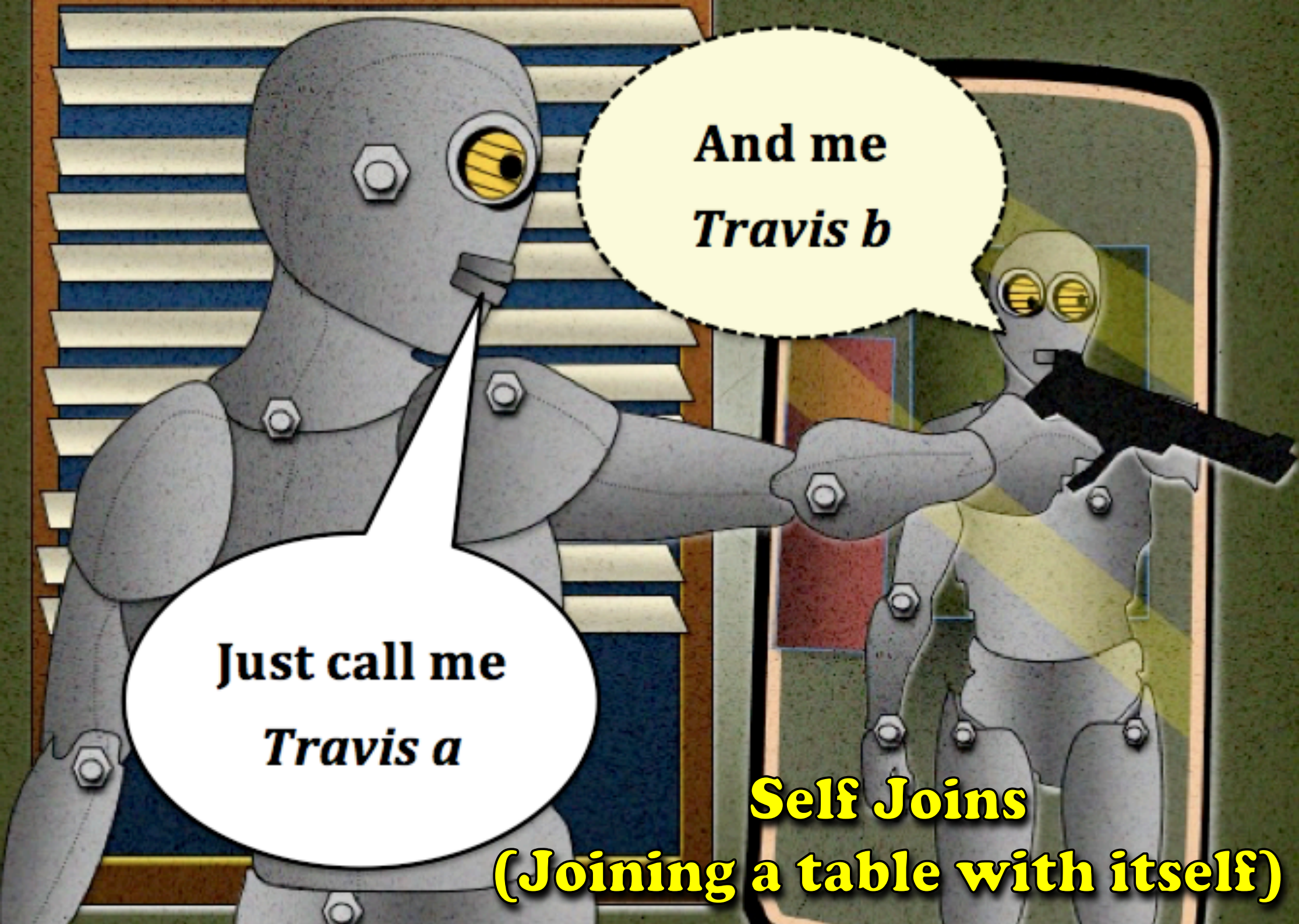
Card_Name	Face	Type	Suit	Face_Value	Game_Value
1H	no	A	H	1	14
2H	no	2	H	2	2

Quantitative Queries on Poker_Hands will require a Join (or two) with Playing_Cards

```
SELECT Player_ID, Game_ID, R1, R2, R3, R4, R5
FROM Poker_Hands, Playing_Cards
WHERE
Poker_Hands.R1 = Playing_Cards.Card_Name
AND Playing_Cards.Face = "yes";
```

Only those hands whose top card is a face card

Player_ID	Game_ID	R1	R2	R3	R4	R5
12789	17MET	QH	QS	3D	3C	3H
12789	23WSA	KH	10H	7H	3H	1H
56347	65YOB	QH	JH	8C	8S	8D


Just call me
Travis a
And me
Travis b
Self Joins
(Joining a table with itself)

```
SELECT Player_ID, Game_ID, R1, R2
FROM
Poker_Hands, Playing_Cards a, Playing_Cards b
WHERE Poker_Hands.R1 = a.Card_Name
AND Poker_Hands.R2 = b.Card_Name
AND a.Face = "yes" AND b.Face = "yes";
```

Player_ID	Game_ID	R1	R2
12789	17MET	QH	QS
56347	65YOB	QH	JH

Hands whose top two cards are face cards

We can join multiple versions of the same table, provided we give each version a unique handle (like a and b)

Regular Joins are sometimes called INNER JOINs

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers
INNER JOIN Orders
ON Customers.Customer_ID = Orders.Customer_ID;
```

Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Leia Organa	Bun-shaped hair curlers	2

Notice the alternative syntax to WHERE constraint

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers
LEFT JOIN Orders
ON Customers.Customer_ID = Orders.Customer_ID;
```

Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Leia Organa	Bun-shaped hair curlers	2
Emperor Palpatine		
Bib Fortuna		

When no matching rows in right table, return null

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers
RIGHT JOIN Orders
ON Customers.Customer_ID = Orders.Customer_ID;
```

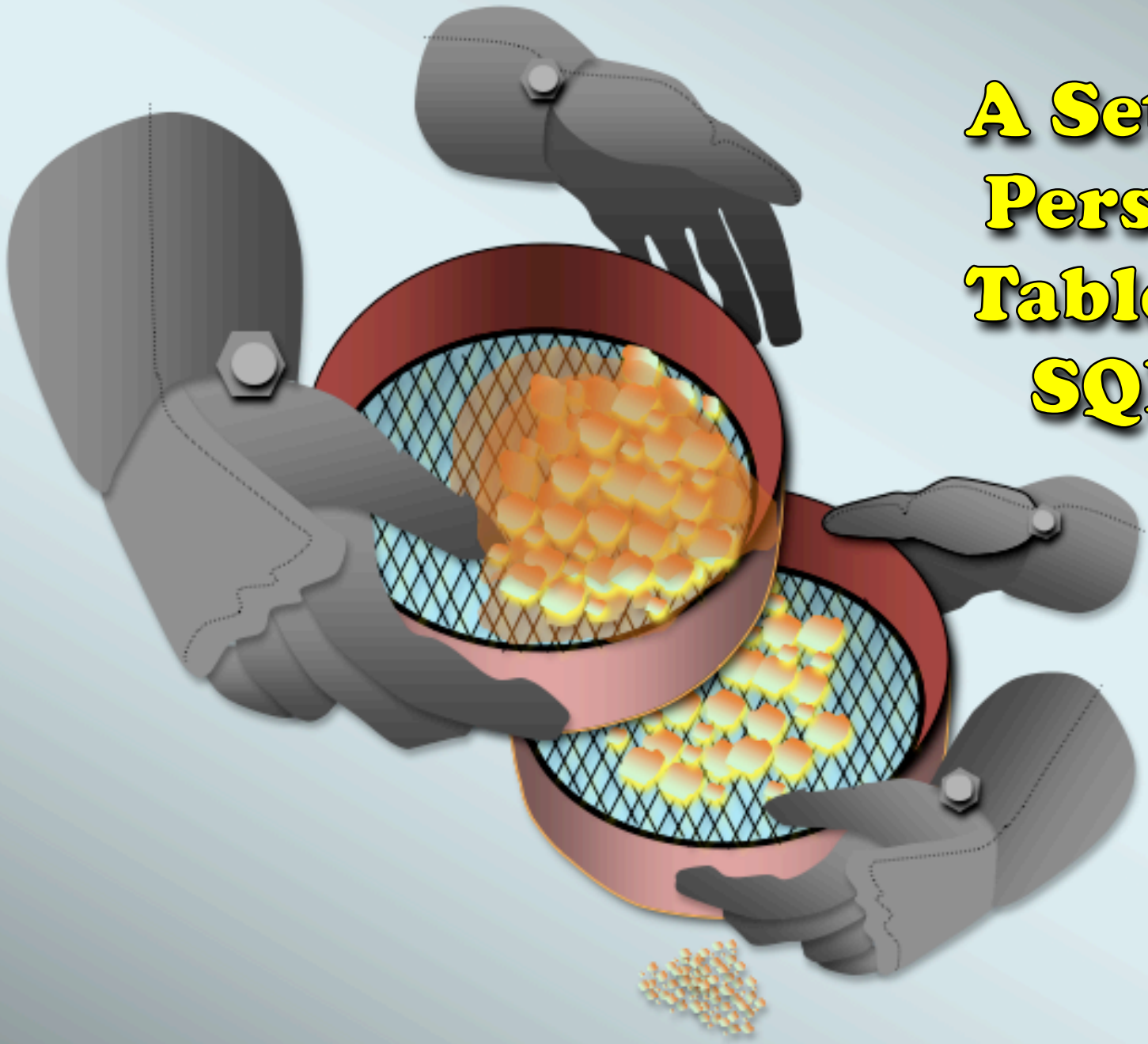
Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Leia Organa	Bun-shaped hair curlers	2
	Sand-speeder model SUX	1
	Gas-powered Harpoon gun	2

When no matching rows in left table, return null

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers
FULL OUTER JOIN Orders
ON Customers.Customer_ID = Orders.Customer_ID;
```

Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Leia Organa	*Bun-shaped hair curlers*	2
	Sand-speeder model SUX	1
	Gas-powered Harpoon gun	2
Sheev Palpatine		
Bib Fortuna		

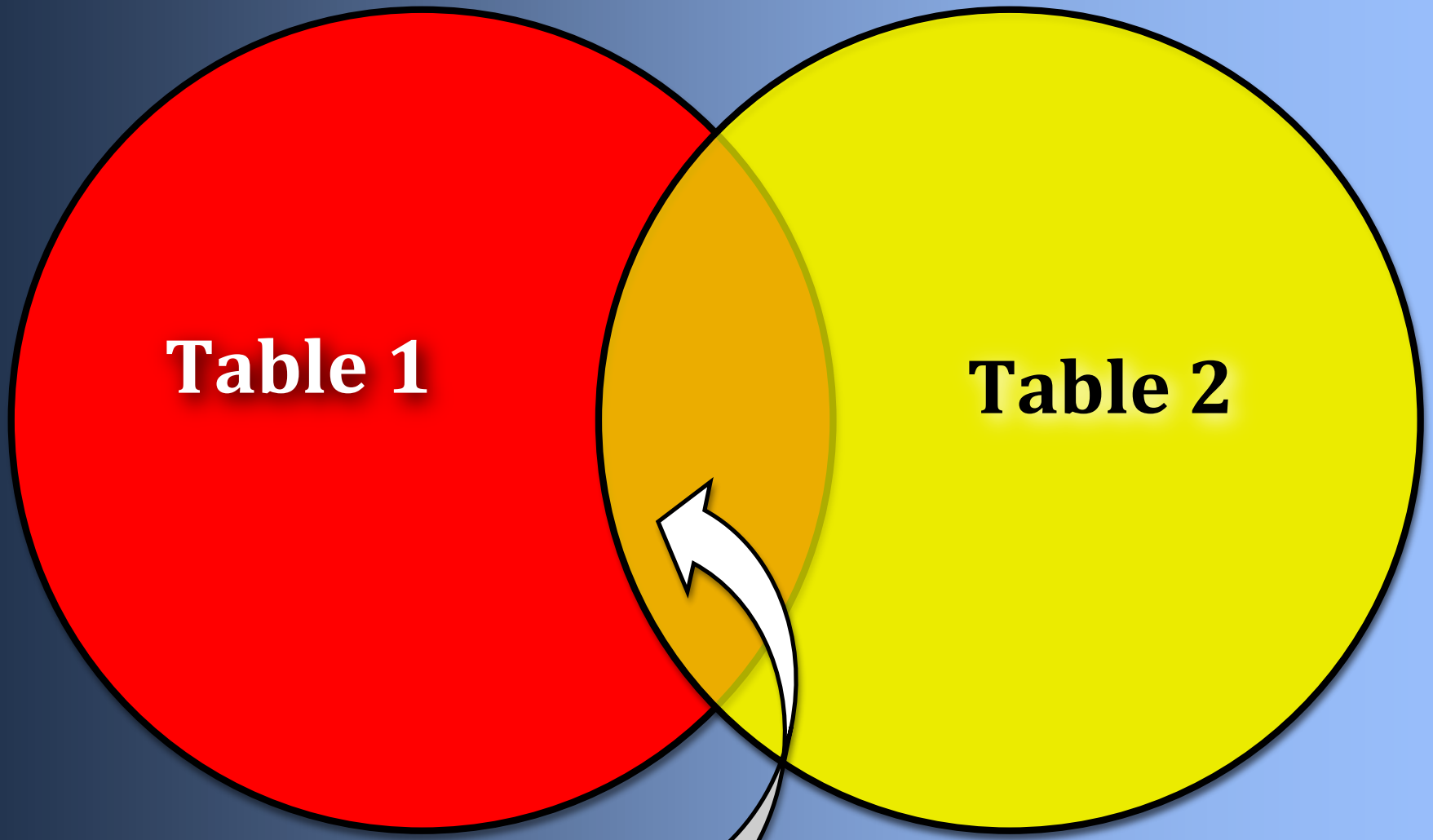
When no matching rows in either table, return null



A Set-Theoretic Perspective on Table Joins and SQL Queries

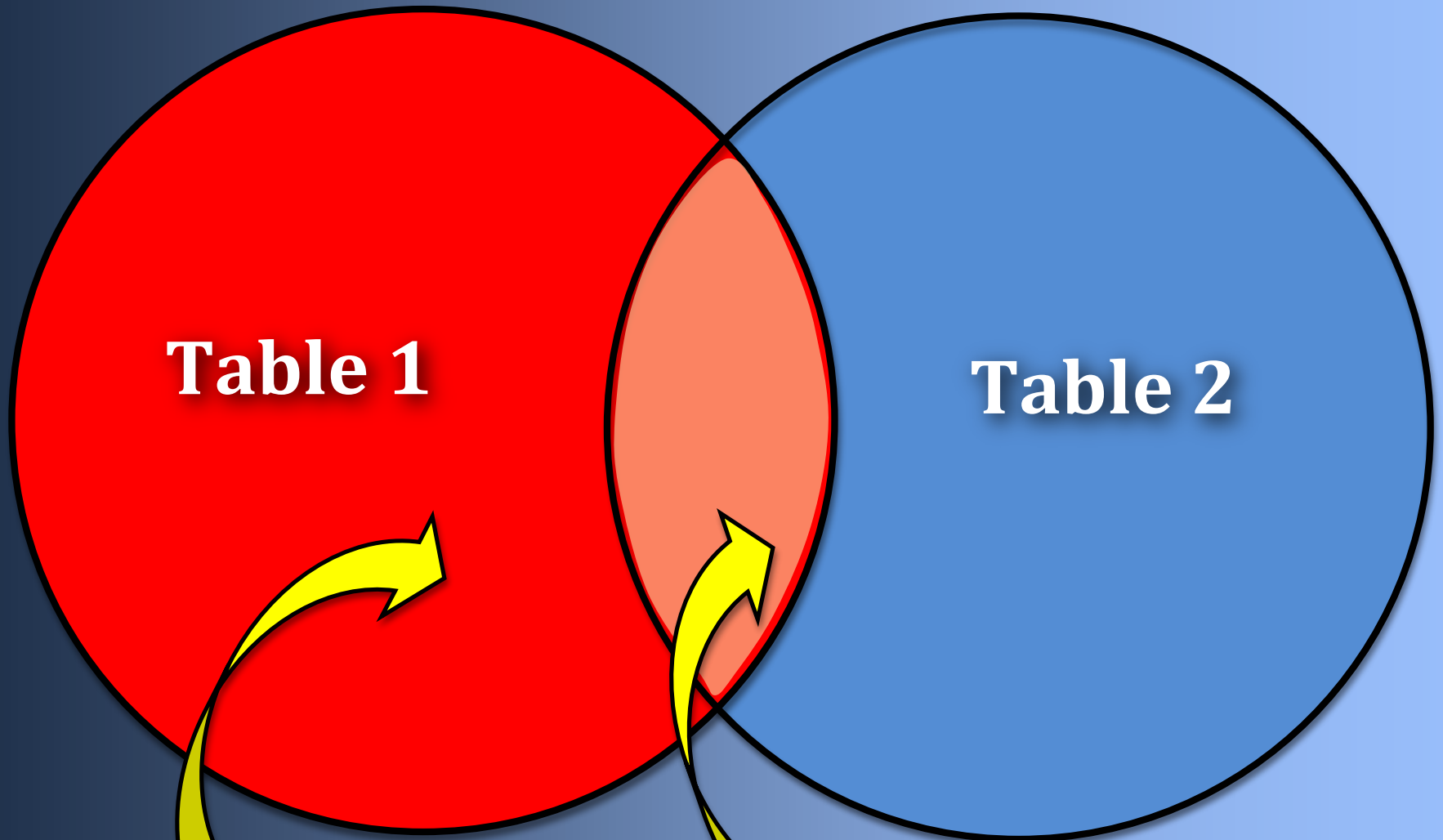
Queries sift information from data

Regular (or INNER) Joins



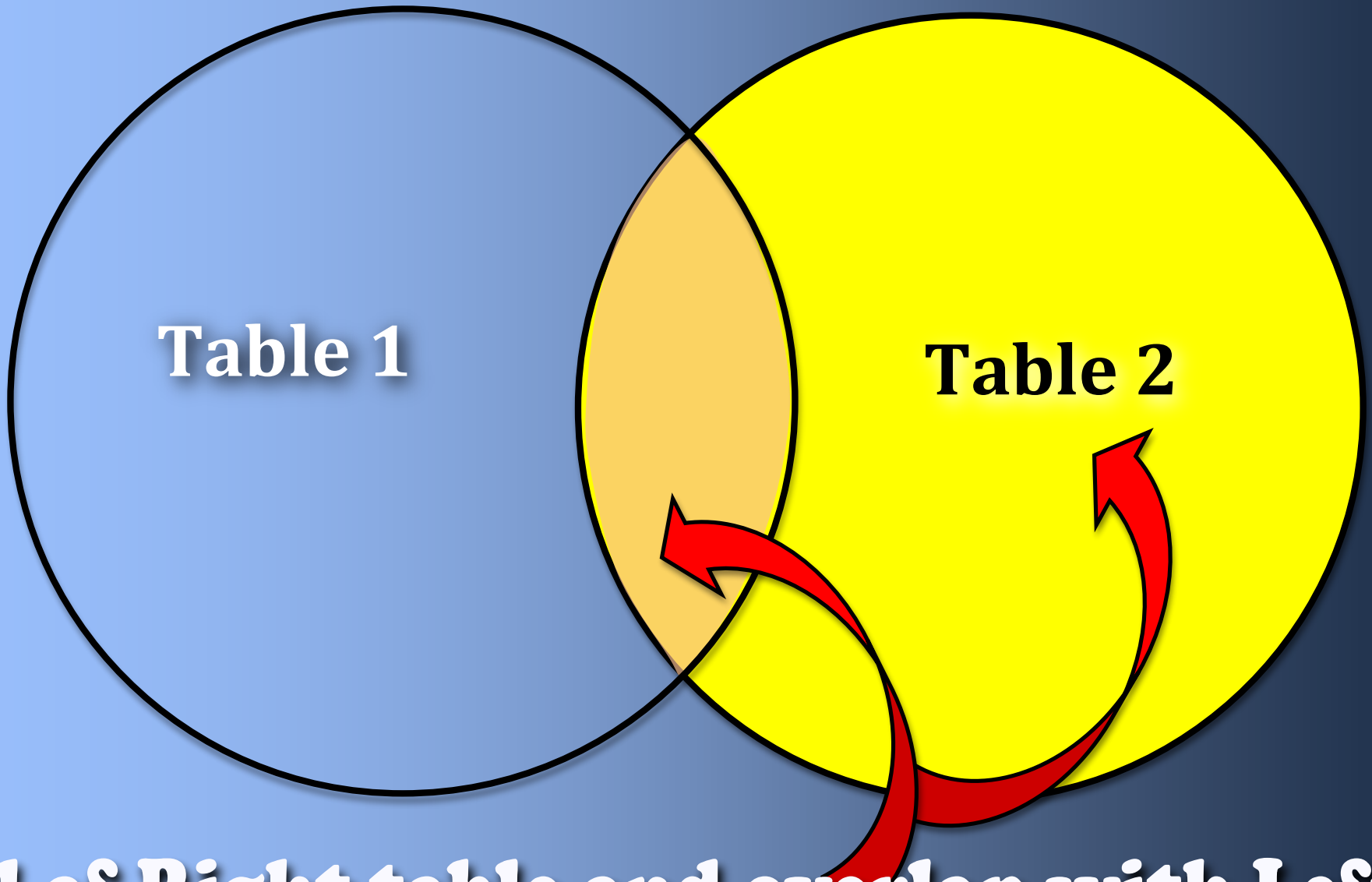
WHERE (or ON) constrained overlap

LEFT (or LEFT OUTER) Joins



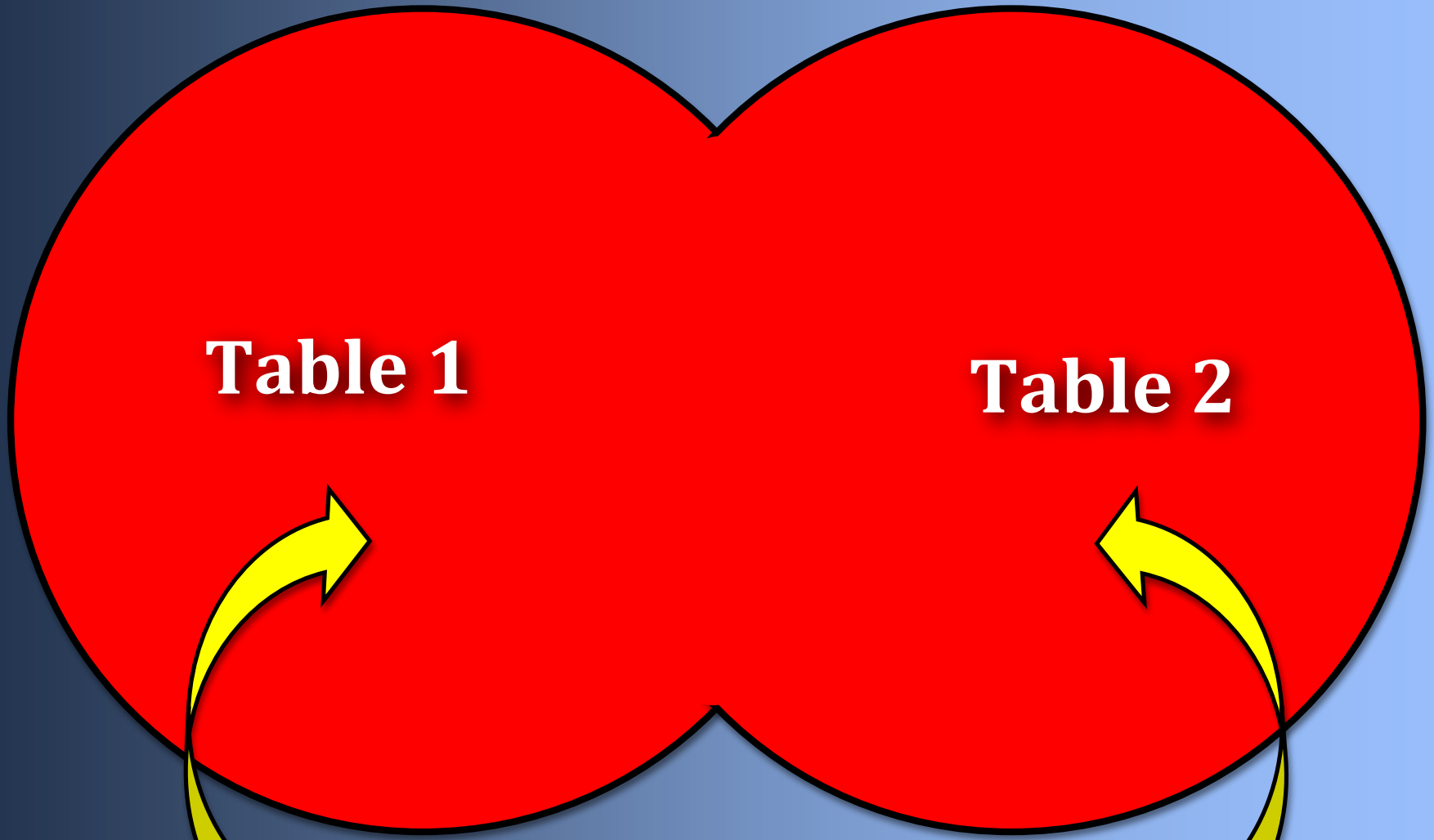
All of Left table and overlap with Right

RIGHT (or RIGHT OUTER) Joins



All of Right table and overlap with Left

FULL OUTER JOINS

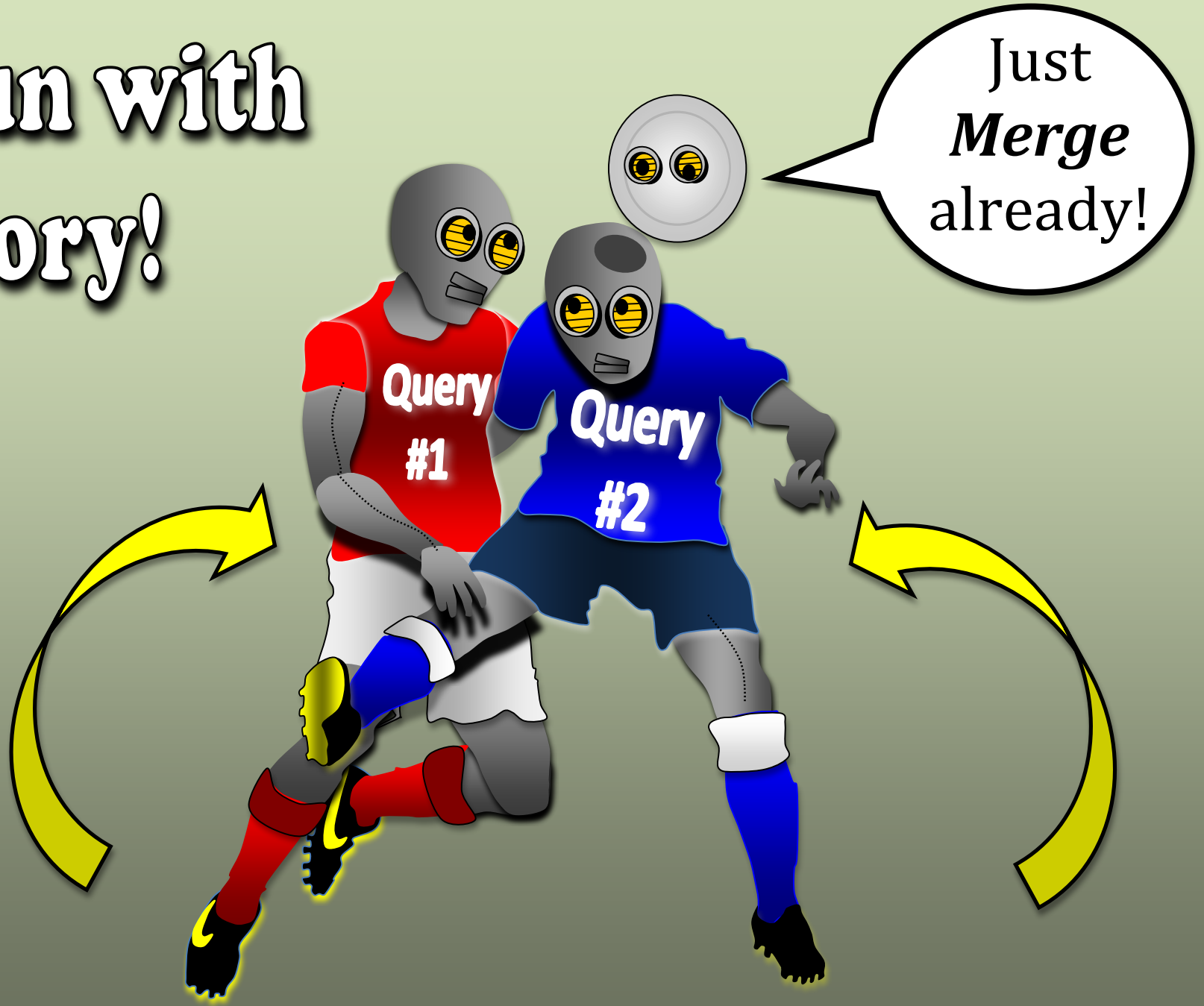


All of LEFT table and All of RIGHT table

Normalization (1NF...3NF) encourages us to factorize our data into the least redundant table structures

Table "joins" are a way of putting the pieces back together again for the purposes of querying across multiple tables and views on the data

More fun with Set theory!



Why Join tables when we can Merge queries?

Let's add another table first ...

Customers (sample rows)

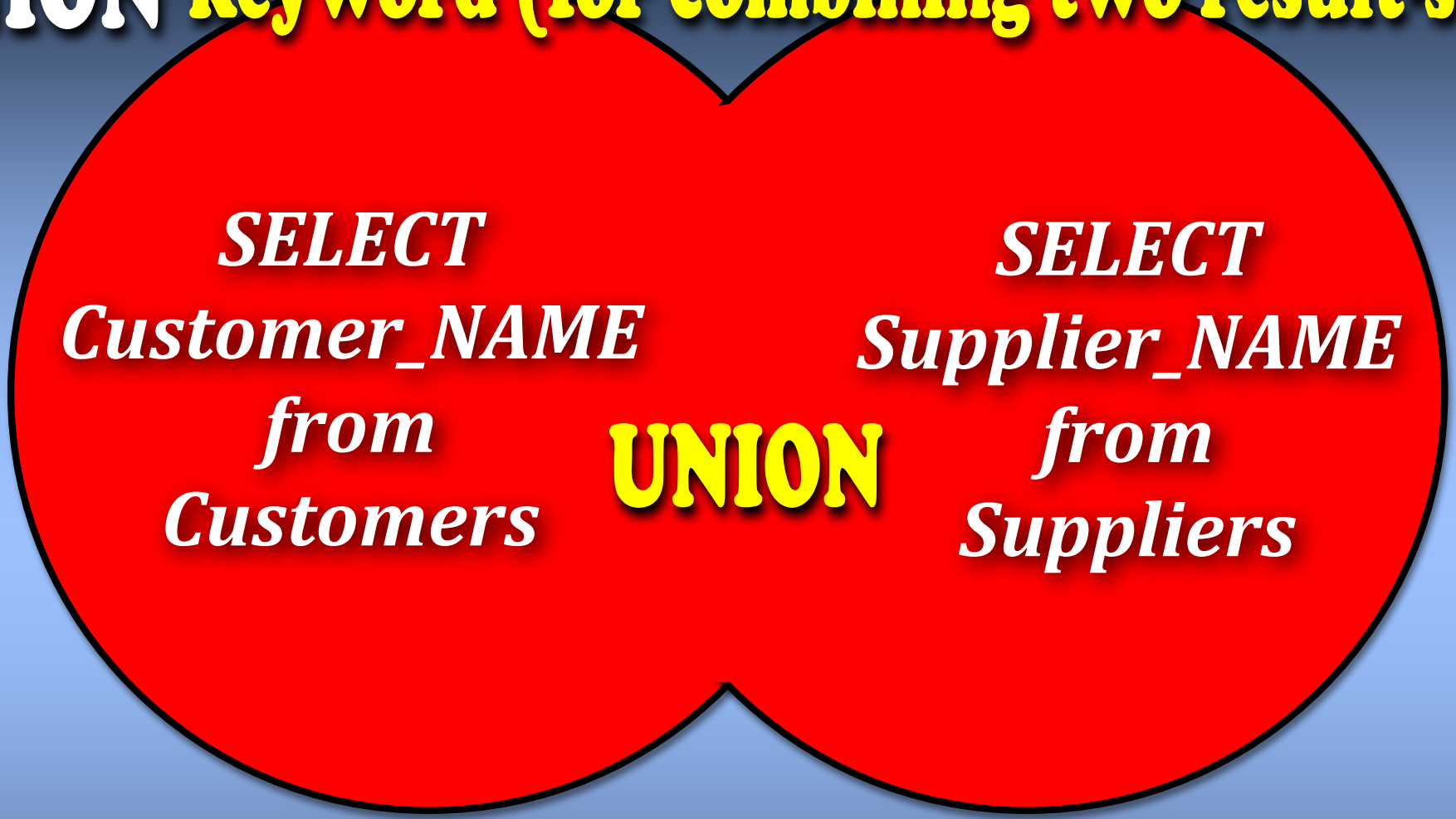
Customer_ID	Customer_NAME	DOB	ADDRESS
THX1138	Darth Vader	28.02.0024	17 Sith Plaza, Tatooine
PLO1001	Leia Organa	14.02.0045	23 Royal Lane, Aldebaran

Suppliers (sample rows)

Supplier_ID	Supplier_NAME	VAT_NUM	ADDRESS
JTH89369	Jabba the Hutt	56347381	29 Slime Court, Tatooine
BF903735	Boba Fett	73735631	16 Sarlacc Pit, Tatooine

Now let's merge queries on different tables

UNION keyword (for combining two result-sets)




```
SELECT Customer_NAME from Customers
UNION
SELECT Supplier_NAME from Suppliers;
```

Comparing Strings with the like constraint

```
SELECT Customer_NAME from Customers
WHERE
Address like "%Tatooine%";
```

```
SELECT Customer_NAME from Customers
WHERE
Address  like  "%Tatooine%"
UNION
SELECT Supplier_NAME from Suppliers
WHERE
Address  like  "%Tatooine%";
```

Returns the unified result-set

Darth Vader, Jabba the Hutt and Boba Fett

Combining Select and Insert statements

```
SELECT *
INTO Local_Customers
FROM Customers
WHERE Address like "%Tatooine%";
```

```
SELECT *
INTO Offworld_Customers
FROM Customers
WHERE Address not like "%Tatooine%";
```

```
SELECT Customer_NAME, ORDER, Quantity
FROM Customers, Orders
WHERE
Customers.Customer_ID = Orders.Customer_ID
ORDER BY Quantity, Customer_NAME;
```

Customer_NAME	ORDER	Quantity
Darth Vader	*Light Sabre, model 27B/6*	3
Boba Fett	*Gas-powered Harpoon gun*	2
Chewbacca	*Wookiee Dandruff shampoo*	2
Leia Organa	*Bun-shaped hair curlers*	2
Biggs Darklighter	*Sand-speeder model SUX*	1

ORDER BY sorts a result-set by given column values

AGGREGATE FUNCTIONS
on Table columns

MIN return smallest value

MAX return largest value

SUM return cumulative value

AVG return mean value

COUNT return num values

COUNT(*) return num rows

What is the total number of products ordered from Tatooine addresses?

```
SELECT SUM(Quantity)
FROM Orders
WHERE Address like "%Tatooine%";
```

What is the average size of an order placed from offworld (outside Tatooine)?

```
SELECT AVG(Quantity)
FROM Orders
WHERE Address not like "%Tatooine%";
```

Let's create a new Employees table:

Staff_ID	Staff_NAME	Salary	Department
THX1138	Darth Vader	256,000	*Human Resources*
REX2910	Sheev Palpatine	999,666	*Upper Management*
BF903735	Boba Fett	127,000	*Contractors*
GMT5639	Wilhuff Tarkin	384,000	*Upper Management*
OKR6758	Orson Krennic	372,000	*Human Resources*
GOR3679	Galen Erso	176,000	*Contractors*
MAV8368	Maximillian Veers	110,846	*Upper Management*

OK, so our "organization" is the Death Star

What is the highest salary in each department in our organization?

```
SELECT Department, MAX(Salary)
FROM Employees
GROUP BY Department;
```

Department	MAX(Salary)
Human Resources	372,000
Upper Management	999,666
Contractors	176,000

We don't want the overall MAX, but the MAX by department

***GROUP BY* is used with aggregate functions to apply them to *groups* of column values**