GENERAL ASSEMBLY

SQL BOOTCAMP

Utku YILDIRIM, Co-Founder & Chief Product Officer at Prorize

HELLO!

AGENDA

- **Part 1**
  - Introduction to Relational Databases
  - Basic CREATE statements
  - Basic SELECT-FROM-WHERE statements
- **Part 2**
  - Complex conditions in the WHERE clause
  - Data manipulation in SELECT statements
- **Part 3**
  - Grouping and aggregations
- **Part 4**
  - JOIN operations
  - Takeaways

MOTIVATION

Decision making process has been always demanding.
Emerging technologies take the guessing out of equation.

It is easier and cheaper to collect data

- Introduction of loyalty programs and online stores shortened feedback loop
- 1 GB data storage used to cost $10 k back in 1990s vs $ 0.1 today

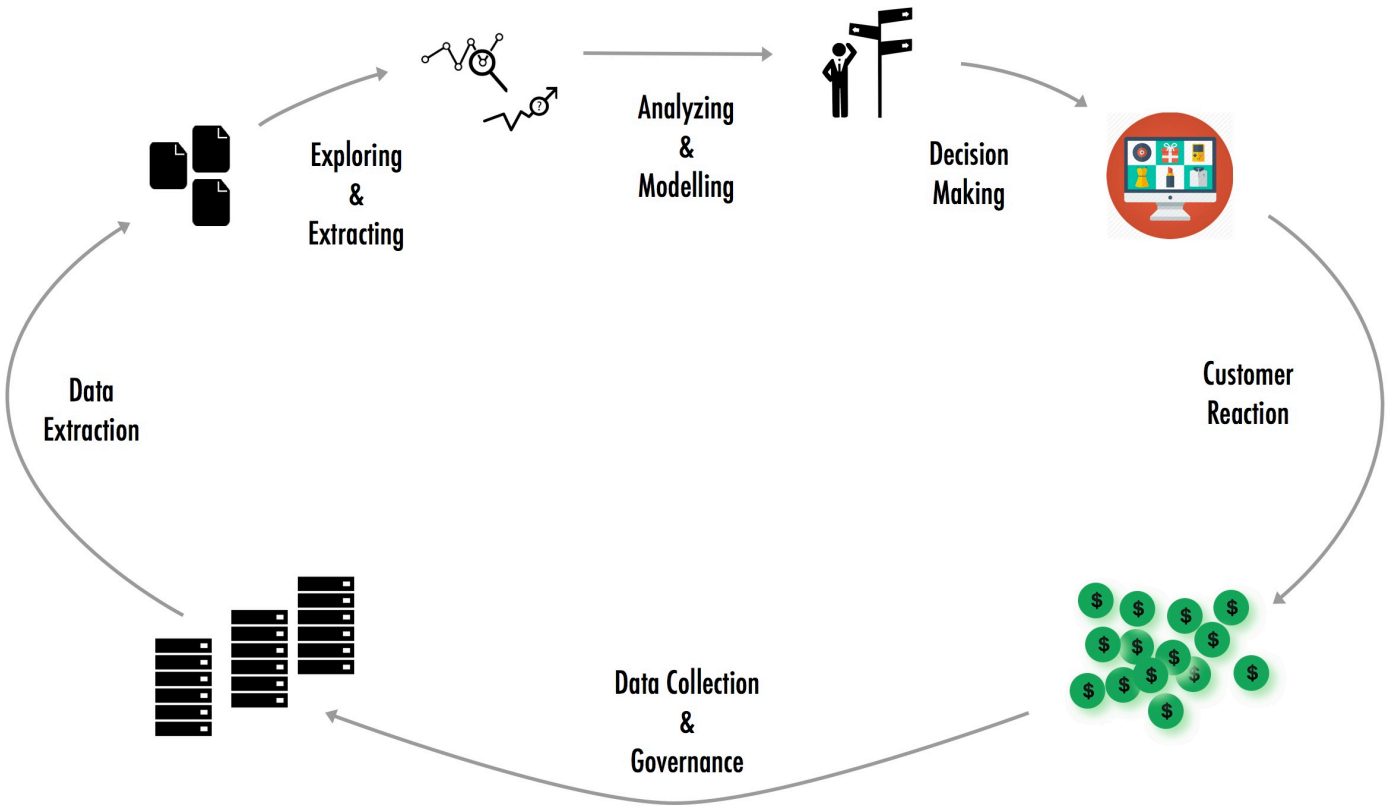
Processing data is easier and faster

- Accessibility and usability of analytical tools improved drastically
- Smartphones today can process 10k more than supercomputers in 1980s

As a result:

- Customer behavior can be identified more accurately as data grows and
- Decisions can be made to quickly adapt to an ever-changing environment

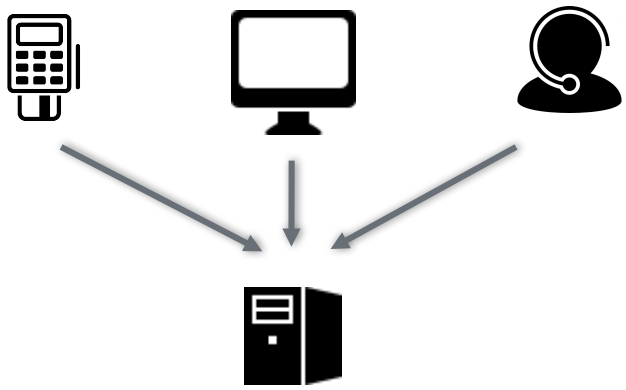
DATA-BACKED DECISION MAKING PROCESS



DATA COLLECTION AND GOVERNANCE

- **What are the considerations?**

  - Volume of data
  - Relationship of entities
  - Limit data types & values
  - User management and security
  - Concurrency
  - Speed of access



HOW DOES RELATIONAL DATABASES WORK?

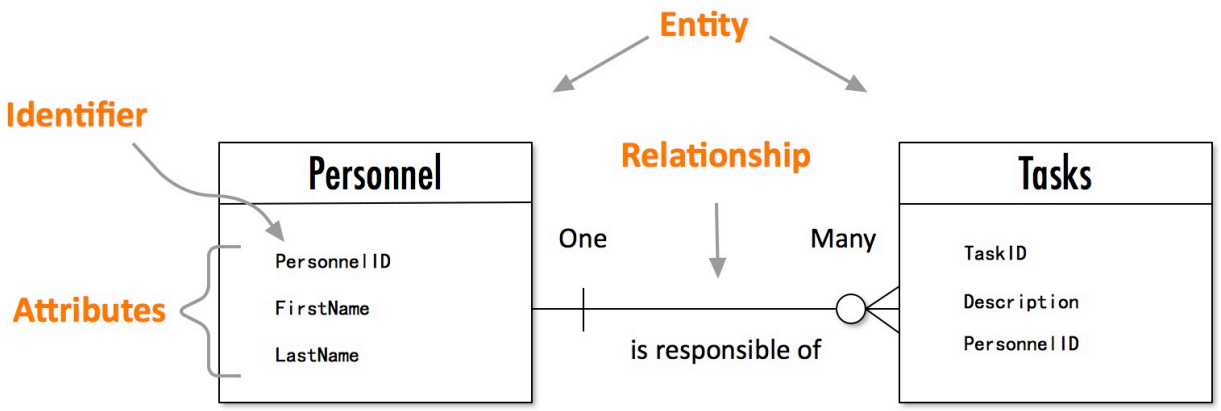
Server-Side

Client-Side

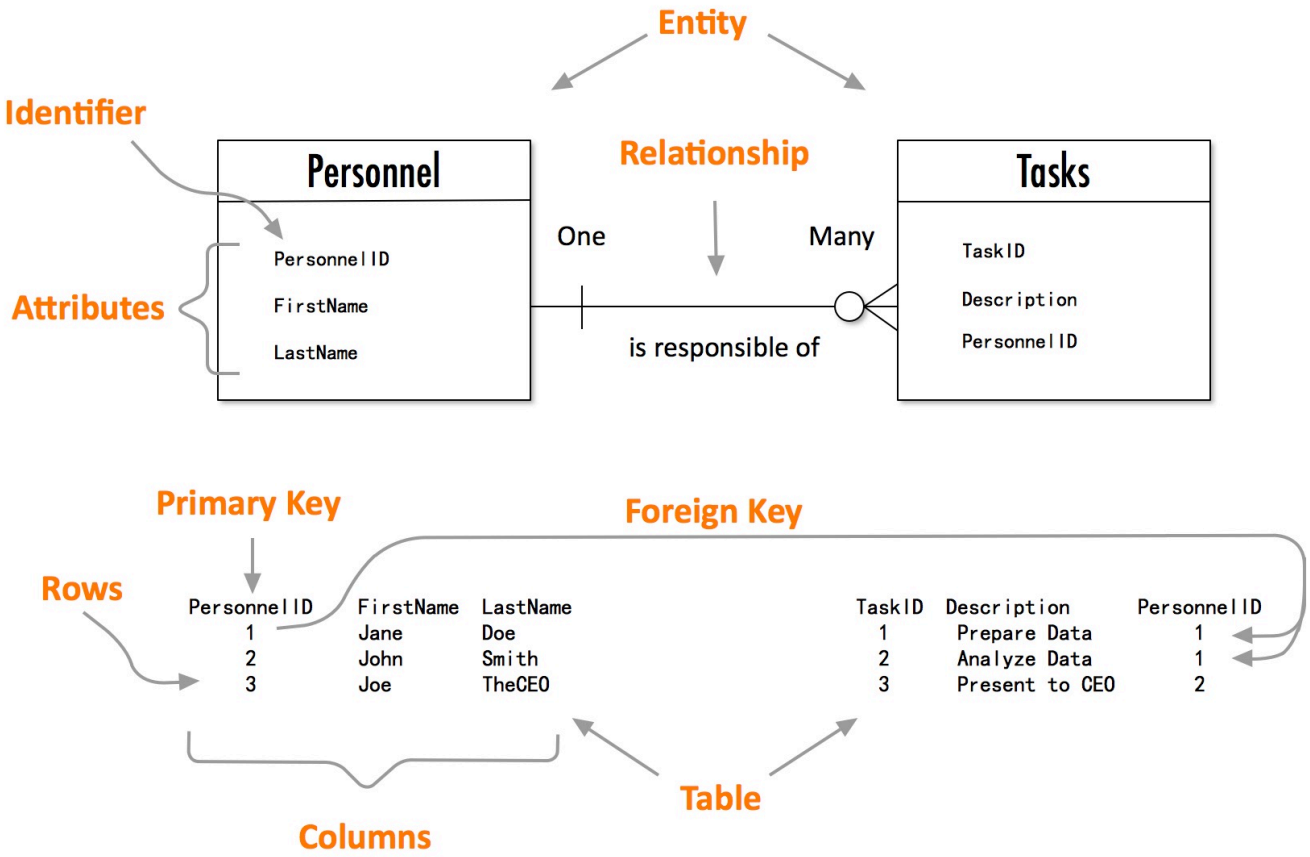
- Think of web servers
- Stores data and manages connections
- Executes instructions

- Think of Chrome, Firefox, Safari, etc.
- Manipulate and retrieve data
- Instructions are created using Structured Query Language

DATABASE ELEMENTS

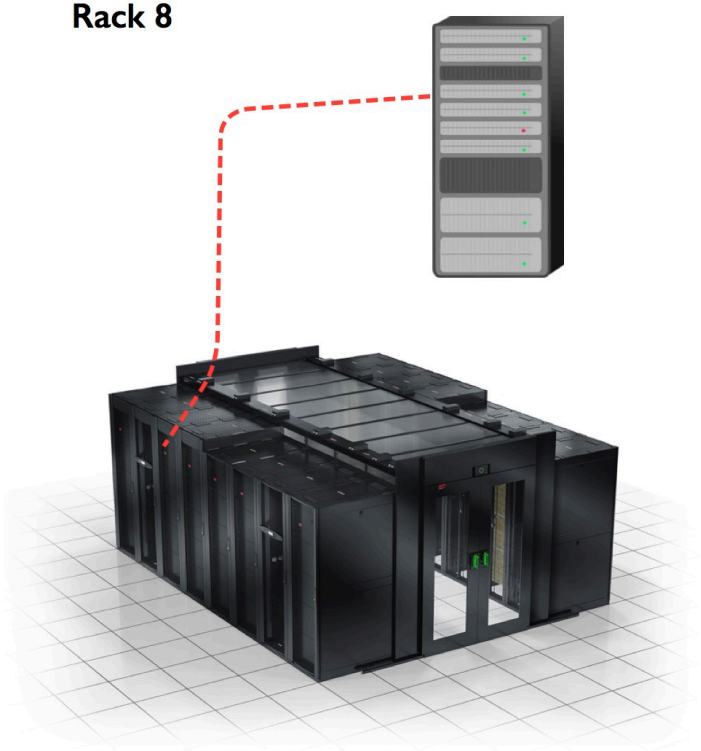


DATABASE ELEMENTS



DEDICATED CLOUD SERVER HOSTING

Rack 8



Units

Unit concept is introduced to identify each physical server and distinguish from other servers in the company. Company might track several metrics for each unit.

Configuration

A configuration represents an item in the company's product portfolio, and plays key role in the prospects decision making process.

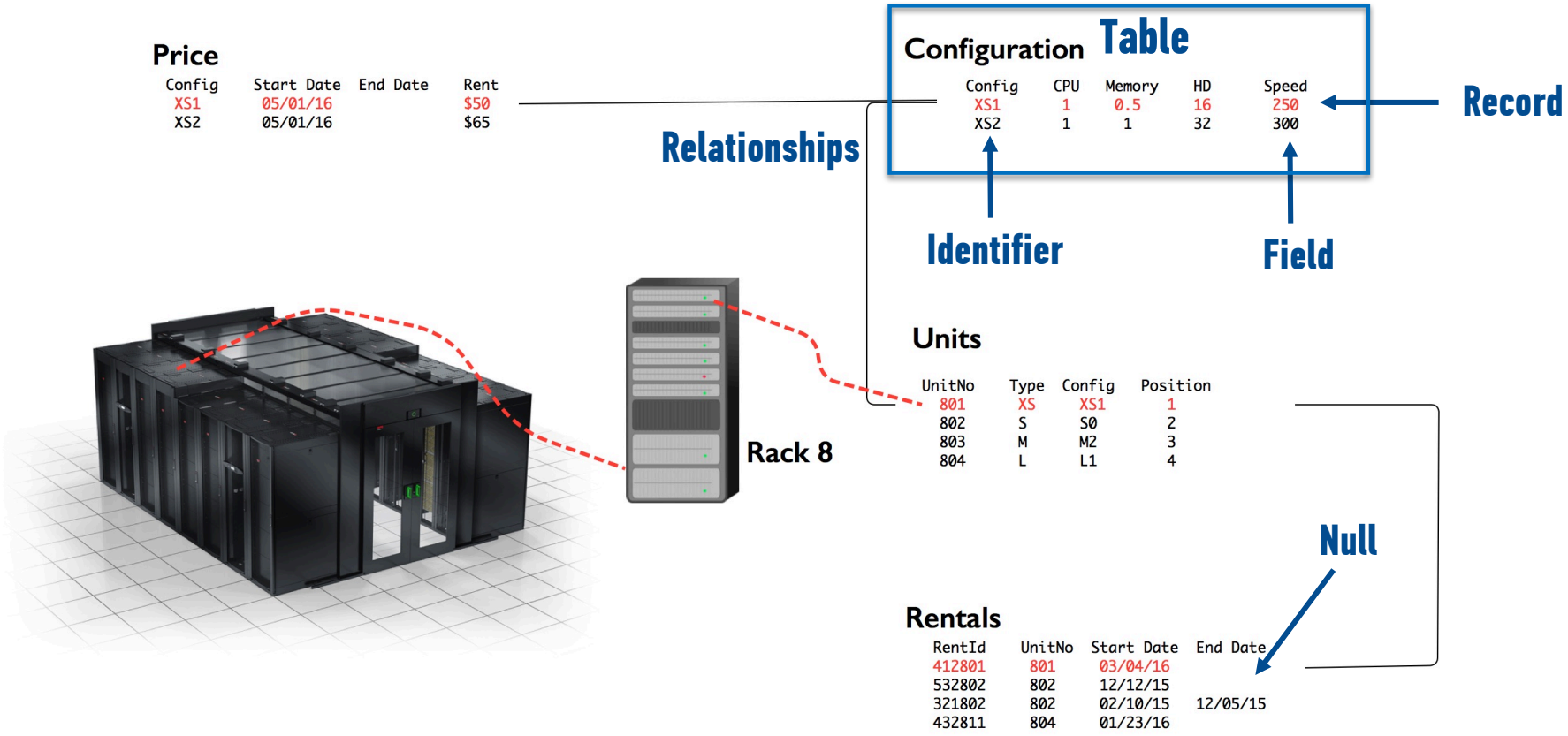
Rentals

Prospects shop around to find the best product and price combination. If a prospect decides to rent, a new rental aggrement is created

Price

Company sets and maintains a price for each configuration depending on quality, availability and competitor pricing.

WHAT IS A RELATIONAL DATABASE?



COMPONENTS OF STRUCTURED QUERY LANGUAGE

Data Definition Language (DDL)

- Business and IT teams get together and define what data needs to be collected and how it will be used
- Create, Alter, Drop, Delete, Truncate

Data Manipulation Language (DML)

- Business team uses DML to manipulate and retrieve data
- Select, Update, Insert

Data Control Language (DCL)

- IT teams decide who can access data and what they can do
- Grant, Revoke, Roles

LET'S CREATE OUR TABLES

Price

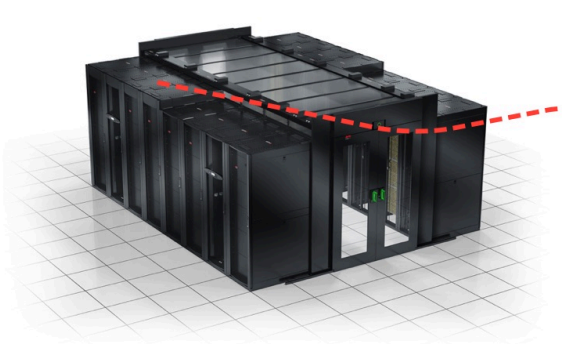
```
create table price
(
    config varchar(3),
    startdate date,
    enddate date,
    rent integer,
    constraint price_configuration_name_fk
    foreign key (config)
    references configuration (config)
);
```

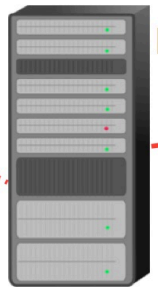
Configuration

```
create table configuration
(
    config varchar(3) primary key not null,
    cpu integer,
    memory integer,
    hd integer,
    speed integer
);
```

Rentals

```
create table rentals
(
    rentid integer primary key not null,
    unitno integer,
    startdate date,
    enddate date,
    constraint rentals_unit_unitno_fk
    foreign key (unitno)
    references unit (unitno)
);
```



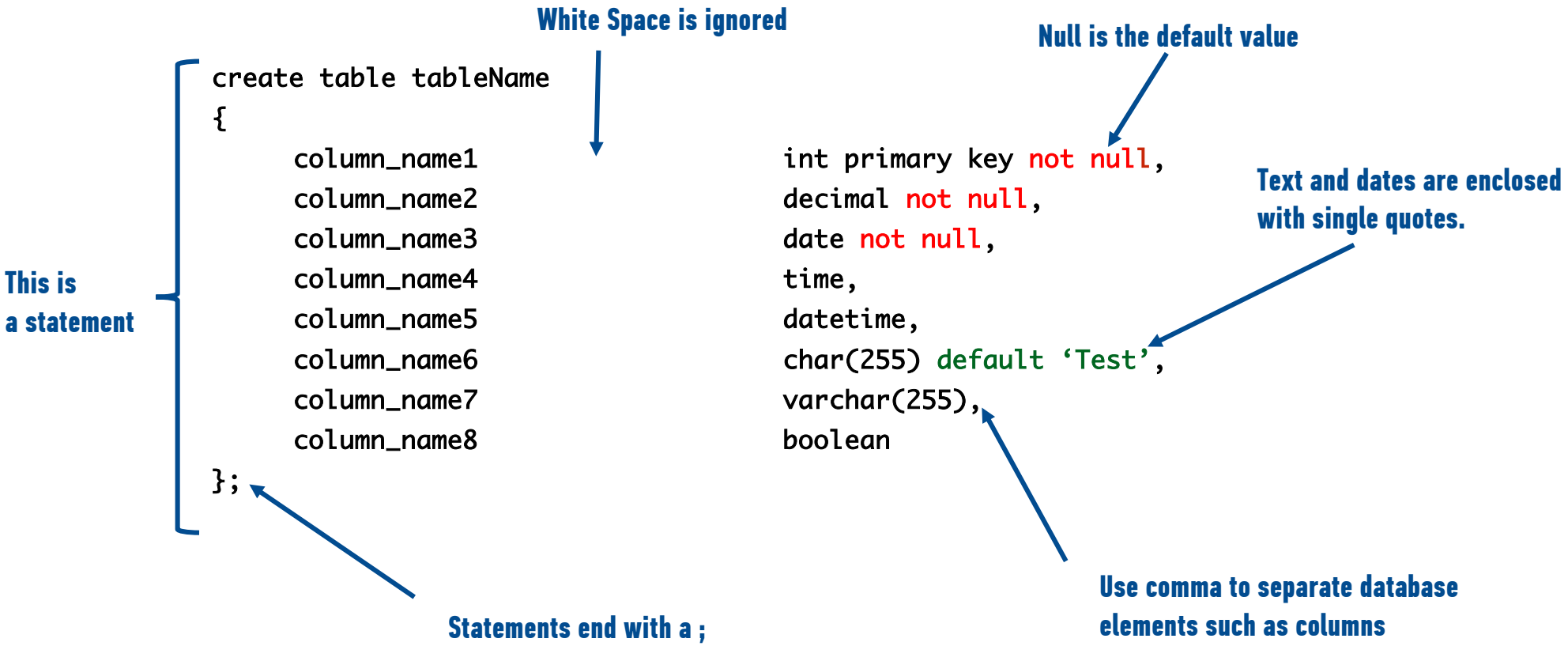


Rack 8

Units

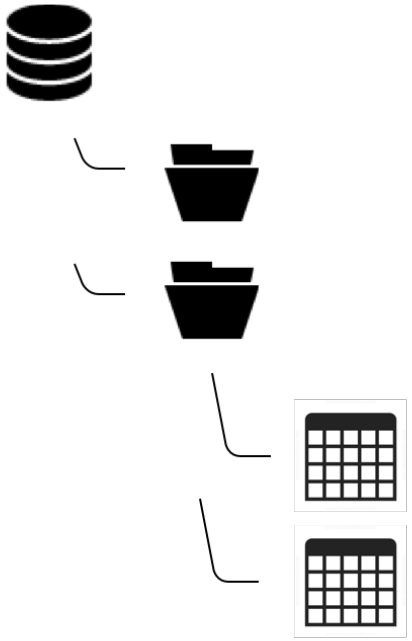
```
create table unit
(
    unitno integer primary key not null,
    type varchar(2),
    config varchar(3),
    position integer,
    constraint unit_configuration_name_fk
    foreign key (config)
    references configuration (config)
);
```

SYNTAX: CREATE A TABLE STATEMENT



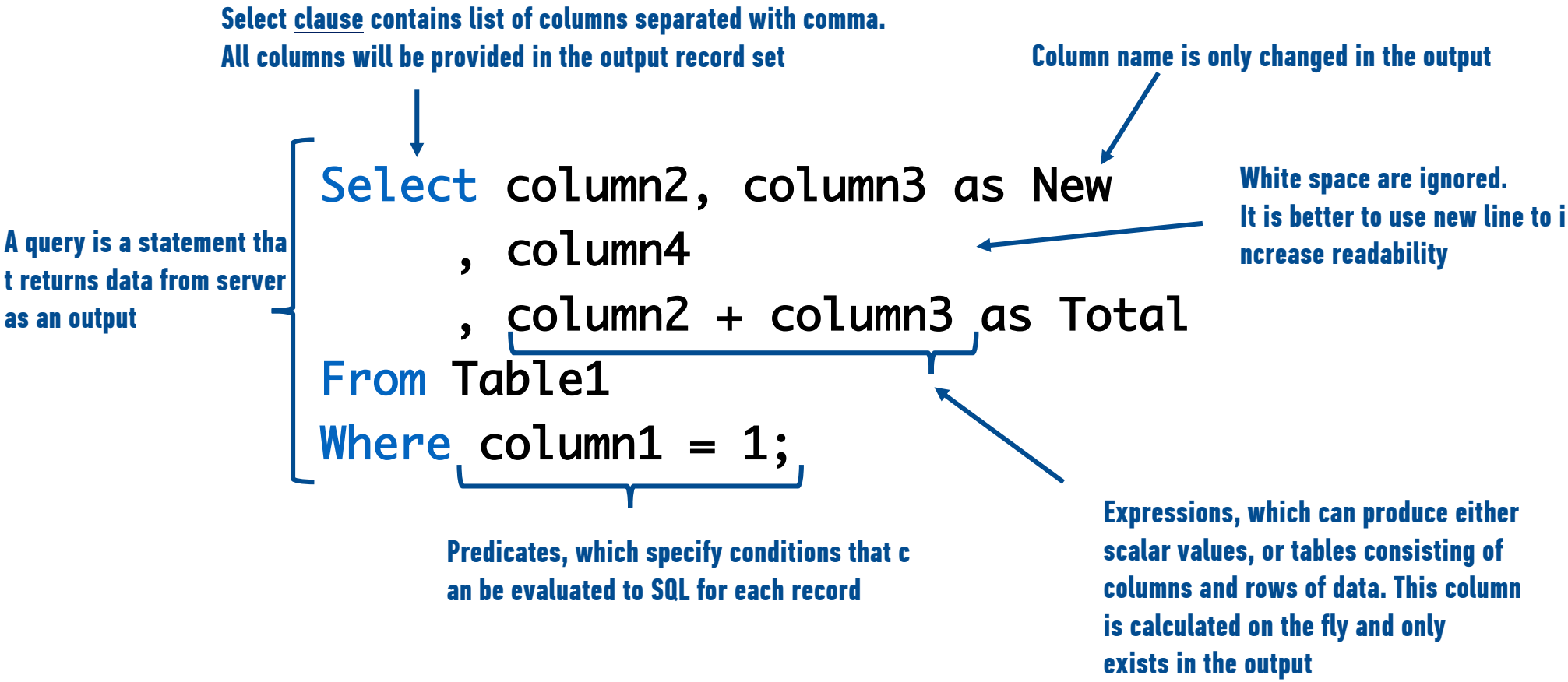

```
create table tableName
{
      column_name1          int primary key not null,
      column_name2          decimal not null,
      column_name3          date not null,
      column_name4          time,
      column_name5          datetime,
      column_name6          char(255) default 'Test',
      column_name7          varchar(255),
      column_name8          boolean
};
```

CREATING DATABASE OBJECTS

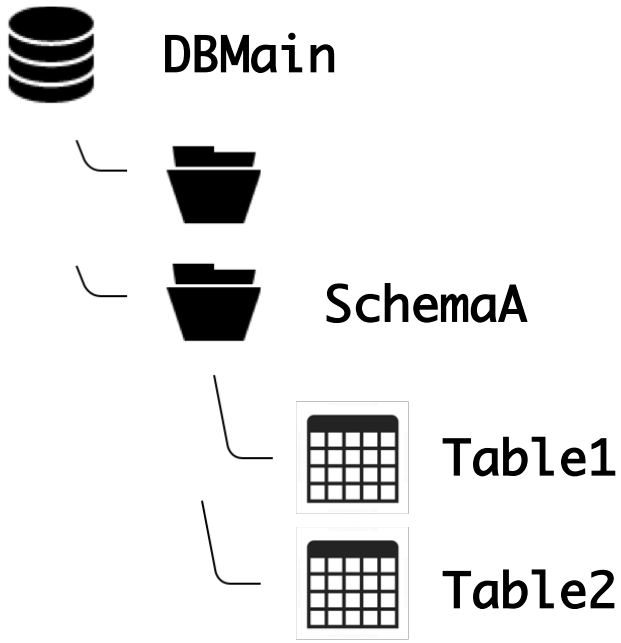


```
create database dbName          drop database dbName

create schema sName             drop schema sName

create table tableName          drop table tableName
{
      column1 int
}
```

SYNTAX: DATA QUERY



ALIASES



Option 1: Fully Qualified Names

```
Select DBMain.SchemaA.Table1.column1
From DBMain.SchemaA.Table1
```

Option 2: Simplified Names and Aliases

```
Use DBMain

Select a.column2, a.column3
From SchemaA.Table1 as a

Select b.* From Table2 as b
```

SYNTAX: FILTERING DATA

Option 1: LIMIT - OFFSET

```
Select a.column2, a.column3
From Table1 as a
Order by column2
Limit {number | ALL} Offset number
```

Option 2: WHERE - DISTINCT

```
Select Distinct b.column4
From Table2 as b
Where b.column1 = 1
```

BRINGING ALL TOGETHER

Clauses needs to follow this order

```
Select Distinct a.column2, a.column3
From Table1 as a
Where column1 = 1
Order by column2
Limit {number | ALL} Offset number
```

PRACTICE 1

1 - 5th sale of the January 2015

2 - Distinct configurations sold less than $100

3 - Top three fastest configurations

4 - Cost of third most expensive unit in ‘M’ Type configurations on January 2016

AGENDA

- **Part 1**
  - Introduction to Relational Databases
  - Basic CREATE statements
  - Basic SELECT-FROM-WHERE statements
- **Part 2**
  - Complex conditions in the WHERE clause
  - Data manipulation in SELECT statements
- **Part 3**
  - Grouping and aggregations
- **Part 4**
  - JOIN operations
  - Takeaways

COMPARISON OPERATORS IN SQL – PART I

```
Where column1 = 1
```

=	equal to
<>, !=	is not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

COMPARISON OPERATORS IN SQL – PART II

IN	In a list of values listed `Color in ('Red', 'Blue', 'White')`
BETWEEN	Between two dates or numbers `ClassDates between '2014-01-01' and '2016-01-01'`
IS NULL	True if value of column is null `EndDate is null`

LIKE	True if
	`'red' LIKE 'red' true`
	`'red' LIKE 'r%' true` → % indicates many characters
	`'red' LIKE '_e_' true` → _ indicates single character
	`'red' LIKE 'd%' false`

ARITHMETIC OPERATORS IN SQL – PART I

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Returns the integer remainder of a division.
^	Exponentiation
@	Absolute value

%, /, * have same priority and are considered before +, -
If %, /, * are used together, priority is set from left to right.

LOGICAL OPERATORS IN SQL

AND	For a row to be selected all the specified conditions must be true.
OR	For the row to be selected at least one of the conditions must be true.
NOT	For a row to be selected the specified condition must be false.

AND

Condition 1	Condition 2	SELECT
True	True	True
True	False	False
False	True	False
False	False	False

OR

Condition 1	Condition 2	SELECT
True	True	True
True	False	True
False	True	True
False	False	False

NOT is considered before AND, AND is considered before OR

EXAMPLE - LOGICAL OPERATORS

```
Select Name
From Configuration
WHERE
     1 = 0
AND
     1 = 0
OR
     1 = 1
AND
     1 = 0
AND
     2 = 2
OR
     1 = 1
AND
     2 =2
```

```
Select Name
From Configuration
WHERE
     (1 = 0
AND
     1 = 0)
OR
     (1 = 1
AND
     1 = 0
AND
     2 = 2)
OR
     (1 = 1
AND
     2 =2)
```

EXAMPLE – COMPARISON OPERATORS

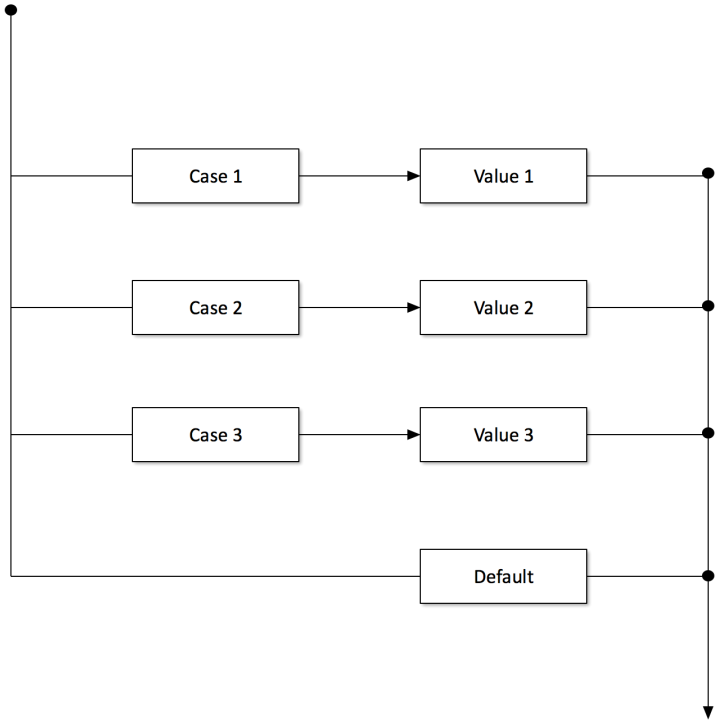
Config	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

- **Configurations that**
  - Have ‘M’ in its name
  - Have speed over 500
  - Memory between 32 and 128
  - Don’t have Memory 64

ARITHMETIC OPERATORS IN SQL – PART II

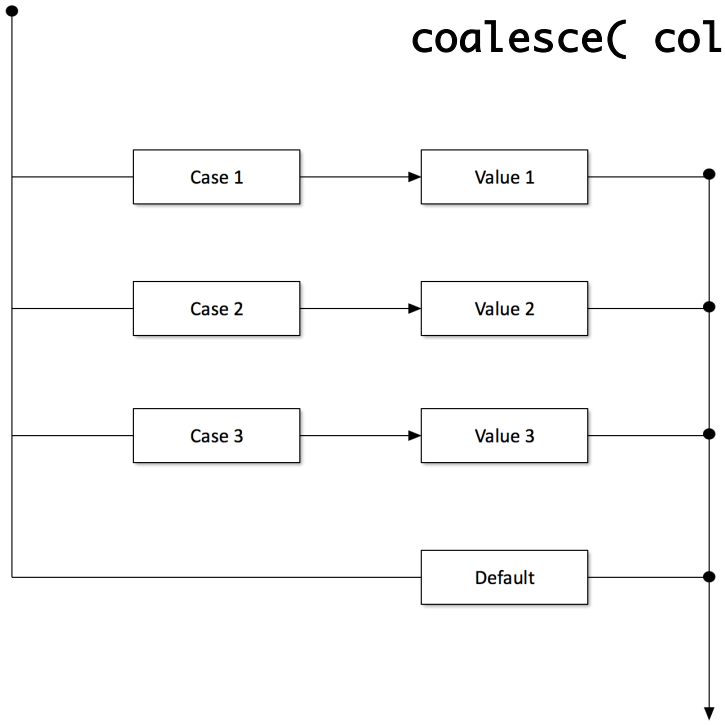
abs(x)	absolute value	abs(-17.4)	17.4
ceil(dp or numeric)	smallest integer not less than argument	ceil(-42.8)	-42
div(y numeric, x numeric)	integer quotient of y/x	div(9,4)	2
exp(dp or numeric)	exponential	exp(1.0)	2.71828182845905
floor(dp or numeric)	largest integer not greater than argument	floor(-42.8)	-43
log(b numeric, x numeric)	logarithm to base b	log(2.0, 64.0)	6.0000000000
power(a numeric, b numeric)	a raised to the power of b	power(9.0, 3.0)	729
round(dp or numeric)	round to nearest integer	round(42.4)	42
round(v numeric, s int)	round to s decimal places	round(42.4382, 2)	42.44
random()	random value in the range 0.0 <= x < 1.0	random()	0.56356

CASE WHEN … THEN …




```
Select column1,

        case when ( case 1 ) then value 1

             when ( case 2 ) then value 2

             when ( case 3 ) then value 3

             else default end

From TableA
```

COALESCE – A SPECIAL CASE




```
coalesce( column2, column3, column4) =

case when column2 is not null then column2
     when column3 is not null then column3
     when column4 is not null then column4
     else null end
```

EXAMPLE – CASE WHEN … THEN …

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

- **Create a HDSizeGroup column and set the value**
  - ‘EntryHD’ if HD < 65
  - ‘MidSizeHD’ if HD > 65 but <300
  - ’EnterpriseHD’ if HD > 300
- **Order by HDSizeGroup**

DATE OPERATORS IN SQL

+	date ‘2016-04-15’ + **integer ‘7’**	date ‘2016-04-22’
+	date ‘2016-04-15’ + **interval ‘7 hours’**	timestamp ‘2016-04-15 07:00:00’
-	time ‘05:00’ – time ‘03:00’	interval ‘02:00:00’
-	date ‘2016-04-22’ – date ‘2016-04-15’	integer ‘7’ days
date_part(text , timestamp)	date_part('hour', timestamp '2016-04-15 20:38:40')	20
date_trunc(text, timestamp)	date_part('hour', timestamp '2016-04-15 20:38:40')	timestamp '2016-04-15 20:00:00'
extract(field from timestamp)	extract(hour from timestamp '2016-04-15 20:38:40')	20
make_date(year int, month int, day int)	make_date(2016, 4, 15)	date ‘2016-04-15’
now()	Current date and time	
age(date startdate, date enddate)		

DATE/TIME DATA TYPES IN SQL

Date: The word followed by a date enclosed in single quotes; DATE ‘2006-2-18’

Other formats are allowed but this is the standard.

Time: The word followed by a time enclosed in single quotes; TIME ‘02:34:00’

Seconds and colons are optional.

Timestamp: Both; TIMESTAMP ’2004-10-23 13:20:00’

Interval: A time interval; INTERVAL ‘2 days’, INTERVAL ‘12 hours’

Many times you in PostGreSQL you don't need to type the descriptor.

For example: '2006-2-18', '02:34', etc.

PRACTICE 2

1 For each date in the rental table identify all rentals started at the beginning of a month, end of a month or in the middle of the month.

2 Identify length of total lease for each rental, and if the end date is not set then use the current date to calculate length of lease in months.

AGENDA

- Part 1
  - Introduction to Relational Databases
  - Basic CREATE statements
  - Basic SELECT-FROM-WHERE statements
- Part 2
  - Complex conditions in the WHERE clause
  - Data manipulation in SELECT statements
- **Part 3**
  - Grouping and aggregations
- **Part 4**
  - JOIN operations
  - Takeaways

AGGREGATE FUNCTIONS

- **Aggregate functions compute summaries of data in a table**
  - Most aggregate functions (all except COUNT) work on a single column of numeric data
  - Use an alias to name the result

- **Aggregate functions**
  - COUNT: The number of rows
  - SUM: The sum of the entries in a column
  - AVG: The average entry in a column
  - MIN, MAX: The minimum and maximum entries in a column

AGGREGATE FUNCTIONS

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select count(*) as Cnt
From Configuration
```

Cnt
11

```
Select sum(memory) as TotMem
From Configuration
```

TotMe
265

```
Select max(speed) as TopSpeed
From Configuration
```

TopSpeed
1250

COMBINING AGGREGATE FUNCTIONS

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select max(speed) – min(speed) as SpeedRange
From Configuration
```

SpeedRange
1000

PRACTICE 3

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

1 Find average speed per CPU count

2 Find average speed weighted by Memory

GROUP BY AGGREGATION

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

- **Sometimes we want to apply aggregate functions to groups of rows**
- **Example: Average Memory for each CPU count.**

```
Select columns1, aggregate functions
From tables
Where conditions
Group by columns1
```

GROUP BY AGGREGATION

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select CPU, avg(Memory) as avgMem
From Configuration
Group by CPU
```

CPU	AvgMem
1	1
2	2.33
4	8
6	24
8	64
12	128

GROUP BY AGGREGATION – FILTERING WITH HAVING

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

- **HAVING is like a WHERE clause, except that it applies to the results of a GROUP BY query**
- **It can be used to select groups which satisfy a given condition**

```
Select CPU
      , avg(Memory) as avgMem
From Configuration
Group by CPU
Having avg(Memory) > 4
```

CPU	AvgMem
4	8
6	24
8	64
12	128

AGGREGATE FUNCTIONS

- **WHERE refers to the rows of tables, and so cannot use aggregate functions**
- **HAVING refers to the groups of rows, and so cannot use columns which are not in the GROUP BY**
- **Think of a query being processed as follows:**
  - Tables are combined
  - WHERE clauses
  - GROUP BY and Aggregates
  - Column selection
  - HAVING clauses
  - ORDER BY

EXAMPLE – CASE WHEN … THEN …

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

- **Create a HDSizeGroup column and set the value**
  - ‘EntryHD’ if HD < 65
  - ‘MidSizeHD’ if HD > 65 but <300
  - ’EnterpriseHD’ if HD > 300
- **Group by HDSizeGroup, find min and max speed as well as average memory**

PRACTICE 4

1. Calculate number of new rentals per month

2. Calculate average list price per month

3. Calculate average rental length in months

AGENDA

- Part 1
  - Introduction to Relational Databases
  - Basic CREATE statements
  - Basic SELECT-FROM-WHERE statements
- Part 2
  - Complex conditions in the WHERE clause
  - Data manipulation in SELECT statements
- Part 3
  - Grouping and aggregations
- **Part 4**
  - JOIN operations
  - Takeaways

DATA NORMALIZATION

- **In general, two tables are joined by mapping**
  - Foreign Keys to Primary Keys
  - Foreign Keys to Foreign Keys
- **Why do we need Primary and Foreign Keys?**
  - Data redundancy
  - Data entry issues
- **Primary and Foreign Keys are created during the data normalization phase**
  - Functional dependencies
  - First, Second, and Third Normal Forms

REDUNDANCY AND NORMALIZATION

Redundant data

- A value is redundant if it can be inferred from another column in the table
  - For example, in a given table there is both student id and student name. Student id can be used to identify the name of the student.
- If the information is duplicated in a table, it is hard to update, delete, or add new information into that table
  - For example, if we want to change the name of the student, we have to change in each record

Normalization

- Aims to reduce data redundancy
- Redundancy is removed by introducing dependencies
  - Student name depends on the student id.
- Normal forms are defined in order to remove certain types of dependency

REDUNDANCY AND NORMALIZATION

Redundant data

- A value is redundant if it can be inferred from another column in the table
  - For example, in a given table there is both student id and student name. Student id can be used to identify the name of the student.
- If the information is duplicated in a table, it is hard to update, delete, or add new information into that table
  - For example, if we want to change the name of the student, we have to change in each record

Normalization

- Aims to reduce data redundancy
- Redundancy is removed by introducing dependencies
  - Student name depends on the student id.
- Normal forms are defined in order to remove certain types of dependency

FIRST NORMAL FORM

- **All data values should be atomic**
  - This means that table entries should be single values, not set of concatenated values or composite objects

Instructor	Type	Format	Courses
I1	Full	Weekends	C1
I2	Full	Weekends	C2, C3
I3	Full	Evenings	C4, C5
I4	CWE	Bootcamp	C3
I5	CWE	Workshop	C5, C1

Instructor	Type	Format	Courses
I1	Full	Weekends	C1
I2	Full	Weekends	C2
I2	Full	Weekends	C3
I3	Full	Evenings	C4
I3	Full	Evenings	C5
I4	CWE	Bootcamp	C3
I5	CWE	Bootcamp	C5
I5	CWE	Workshop	C1

FIRST NORMAL FORM - PROBLEMS

- **INSERT anomalies**
  - Can't add an instructor with no courses
- **UPDATE anomalies**
  - To change Format for I2 , we have to change two rows
- **DELETE anomalies**
  - If we remove I3, we remove Evenings as well

Instructor	Type	Format	Courses
I1	Full	Weekends	C1
I2	Full	Weekends	C2
I2	Full	Weekends	C3
I3	Full	Evenings	C4
I3	Full	Evenings	C5
I4	CWE	Bootcamp	C3
I5	CWE	Bootcamp	C5
I5	CWE	Workshop	C1

SECOND NORMAL FORM

- Type and Format depend on Instructor
- Instructor is assigned to the courses

Instructor	Type	Format	Courses
I1	Full	Weekends	C1
I2	Full	Weekends	C2
I2	Full	Weekends	C3
I3	Full	Evenings	C4
I3	Full	Evenings	C5
I4	CWE	Bootcamp	C3
I5	CWE	Bootcamp	C5
I6	CWE	Workshop	C1

Instructor	Type	Format
I1	Full	Weekends
I2	Full	Weekends
I3	Full	Evenings
I4	CWE	Bootcamp
I5	CWE	Workshop

Instructor	Courses
I1	C1
I2	C2
I2	C3
I3	C4
I3	C5
I4	C3
I5	C5
I6	C1

SECOND NORMAL FORM – PROBLEMS

- **INSERT anomalies**
  - Can't add a format with no instructors
- **UPDATE anomalies**
  - To change type of Weekends, we have to change two rows
- **DELETE anomalies**
  - If we remove I4, we remove Bootcamp as well

Instructor	Type	Format
I1	Full	Weekends
I2	Full	Weekends
I3	Full	Evenings
I4	CWE	Bootcamp
I5	CWE	Workshop

THIRD NORMAL FORM

- Type depends on Format

Instructor	Type	Format
I1	Full	Weekends
I2	Full	Weekends
I3	Full	Evenings
I4	CWE	Bootcamp
I5	CWE	Workshop

Type	Format
Full	Weekends
Full	Evenings
CWE	Bootcamp
CWE	Workshop

Instructor	Format
I1	Weekends
I2	Weekends
I3	Evenings
I4	Bootcamp
I5	Workshop

THIRD NORMAL FORM

Instructor	Type	Format	Courses
I1	Full	Weekends	C1
I2	Full	Weekends	C2, C3
I3	Full	Evenings	C4, C5
I4	CWE	Bootcamp	C3
I5	CWE	Workshop	C5, C1

Instructor	Courses
I1	C1
I2	C2
I2	C3
I3	C4
I3	C5
I4	C3
I5	C5
I6	C1

Format	Type
Weekends	Full
Evenings	Full
Bootcamp	CWE
Workshop	CWE

Instructor	Format
I1	Weekends
I2	Weekends
I3	Evenings
I4	Bootcamp
I5	Workshop

JOINS

- **What can we achieve with a join?**
  - Compare two data sources
  - Get more information about an entity
  - Use different tables for calculating a metric

- **Relationship between tables**
  - One-to-One
  - One-to-Many
  - Many-to-Many

CROSS JOINS

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Cross join Table2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

INNER JOINS – ON CONDITION WITH EQUALITY

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Inner join Table2
On Table1.P1 = Table2.P2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

```
Select *
From Table1, Table2
Where Table1.P1 = Table2.P2
```

```
Select *
From Table1
Join Table2
On Table1.P1 = Table2.P2
```

INNER JOINS – ON CONDITION WITH EQUALITY

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Join Table2
On Table1.P1 >=
Table2.P2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

```
Select *
From Table1
Join Table2
On Table1.P1 = Table2.P2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

INNER JOINS – ON CONDITION WITH INEQUALITY

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Join Table2
On Table1.P1 < Table2.P2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

```
Select *
From Table1
Join Table2
On Table1.P1 = Table2.P2
```

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

OUTER JOINS

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Full Outer Join Table2
On Table1.P1 = Table2.P2
```

Inner

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

Outer

P1	T1	P2	T2
1	C1		
2	C2		
3	C3		
		2	C3
		3	C1
		4	C2

LEFT (OUTER) JOINS

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Left Outer Join Table2
On Table1.P1 = Table2.P2
```

Inner

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

Left Outer

P1	T1	P2	T2
1	C1		
2	C2		
3	C3		

RIGHT (OUTER) JOINS

Table 1

P1	T1
1	C1
2	C2
3	C3

P2	T2
2	C3
3	C1
4	C2

Table 2

```
Select *
From Table1
Right Outer Join Table2
On Table1.P1 = Table2.P2
```

Inner

P1	T1	P2	T2
1	C1	2	C3
2	C2	2	C3
3	C3	2	C3
1	C1	3	C1
2	C2	3	C1
3	C3	3	C1
1	C1	4	C2
2	C2	4	C2
3	C3	4	C2

Right Outer

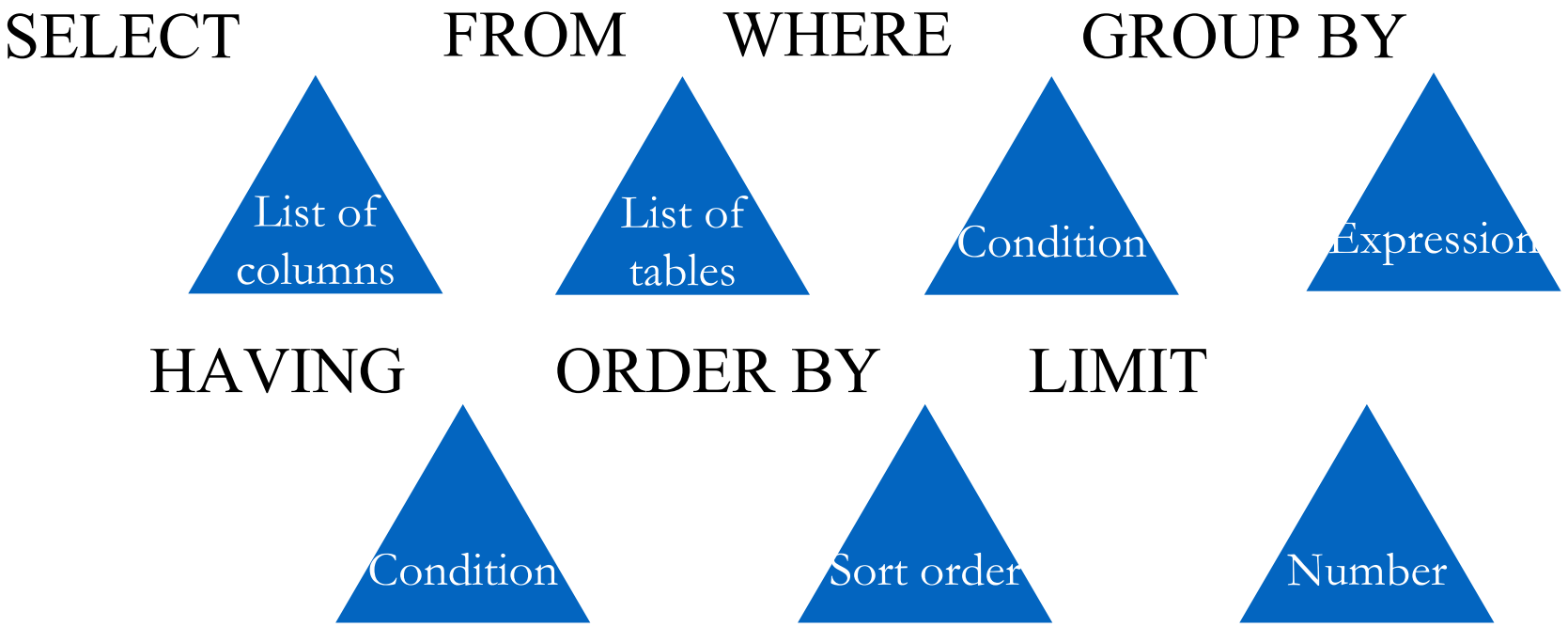
P1	T1	P2	T2
		2	C3
		3	C1
		4	C2

PRACTICE 5

1 Calculate total number of available units at the beginning of January 2016

2 Calculate average listed rental price that new rentals paid after September 2015

ORDER OF CLAUSES



SELF JOINS – ON CONDITION WITH INEQUALITY

Table 1

P1	T1
1	C1
2	C2
3	C3

P1	T1
1	C1
2	C2
3	C3

Table 1

```
Select *
From Table1 as a
Join Table1 as b
On a.P1 < b.P1
```

P1	T1	P1	T1
1	C1	1	C1
2	C2	1	C1
3	C3	1	C1
1	C1	2	C2
2	C2	2	C2
3	C3	2	C2
1	C1	3	C3
2	C2	3	C3
3	C3	3	C3

PRACTICE 6

Create a new team using the players who have larger salaries than their captain, and find the total earning of the new team you created.

```
create table Players (
	PlayerID int not null,
	NBATeamID int not null,
	CaptainID int null,
	Salary int not null,
	Name varchar(100) not null,
	Primary key (PlayerID),
	constraint m_captain
		foreign key (CaptainID)
		references Players (PlayerID)
);
create table Teams (
	NBATeamID int not null,
	Name varchar(50)
);
```

```
INSERT INTO Players
(PlayerId, NBATeamID, CaptainID, Salary, Name)
VALUES (1, 1, 2, 150, 'A'), (2, 1, 2, 100, 'B'),
(3, 1, 2, 124, 'C'), (4, 1, 2, 90, 'D'),
(5, 1, 2, 85, 'E'), (6, 2, 8, 123, 'F'),
(7, 2, 8, 250, 'G'), (8, 2, 8, 150, 'H'),
(9, 2, 8, 175, 'I'), (10, 2, 8, 75, 'J')
;
INSERT INTO Teams
(NBATeamId, Name)
VALUES (1, 'Atlanta Hawks'), (2, 'Miami Heat')
;
```

BONUS

SUBQUERIES

- Find the third fastest configuration

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select *
From Configuration
Where Speed = ( Select min(Speed)
                From (Select *
                      From Configuration
                      Order by Speed Desc
                      Limit 3) as a)
```

SUBQUERIES

- **Find all the configurations that have a price higher than $150 this month**

Name	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select *
From Configuration
Where Config in ( select Config
                  from Price
                  where StartDate >= '01-01-2016'
                    and Rent > 150)
```

SUBQUERIES

- Find the average speed of Small Servers

Config	CPU	Memory	HD	Speed
XS1	1	1	16	250
XS2	1	1	32	300
S1	2	1	64	325
S2	2	2	64	375
S3	2	4	128	450
M1	4	8	64	550
M2	4	8	64	650
M3	6	16	256	750
M4	6	32	512	850
L1	8	64	1024	950
L2	12	128	2048	1250

```
Select avg(Speed)
From ( select *
       from Configuration
       where Config like 'S%') as a
```

PRACTICE 7

1 Find the configuration that has a Memory closest to the average Memory.

2 Identify the configurations that sold more than 10 units in first three month of 2016

FLASHBACK : CREATE A TABLE

SQL data type is an attribute that specifies type of data.

```
create table tableName
{
     column_name1                    int v,
     column_name2                    decimal not null,
     column_name3                    date not null,
     column_name4                    time,
     column_name5                    datetime,
     column_name6                    char(255) default ‘Test’,
     column_name7                    varchar(255),
     column_name8                    boolean,
     PRIMARY KEY( column_name1)
}
```

INSERT DATA

Option 1:

```
insert tableName (column_name2, column_name3, column_name4, column_name5)
values ( 13.5, '01-01-2016', '12:00:00', '01-01-2016 12:00:00'),
       ( 15.5, '02-01-2016', '12:00:00', '02-01-2016 12:00:00'),
```

Option 2:

```
insert tableName
values ( 13.5,    '01-01-2016', '12:00:00', '01-01-2016 12:00:00','January','Sales',1),
        ( 15.5,   '02-01-2016', '12:00:00', '02-01-2016 12:00:00','February','Sales',0),
```

UPDATE DATA

Option 1: Update all the rows

```
Update tableName
Set Column1 = Column2
```

Option 2: Update only the rows in the join with the second table

```
Update a
Set Column1 = b.Column3
From tableName1 as a, tableName2 as b
Where a.Column2 = b.Column2
```

PRACTICE 8

Insert a new team.
Update the team of the players who have larger salaries than their captain with the new team.
Update their captain as the 3rd most earning player for the new team you created.

```
create table Players (
	PlayerID int not null,
	NBATeamID int not null,
	CaptainID int null,
	Salary int not null,
	Name varchar(100) not null,
	Primary key (PlayerID),
	constraint m_captain
		foreign key (CaptainID)
		references Players (PlayerID)
)
create table Teams (
	NBATeamID int not null,
	Name varchar(50)
)
```

```
INSERT INTO Players
([PlayerId], [NBATeamID], [CaptainID], [Salary], [Name])
VALUES (1, 1, 2, 150, 'A'), (2, 1, 2, 100, 'B'),
(3, 1, 2, 124, 'C'), (4, 1, 2, 90, 'D'),
(5, 1, 2, 85, 'E'), (6, 2, 8, 123, 'F'),
(7, 2, 8, 250, 'G'), (8, 2, 8, 150, 'H'),
(9, 2, 8, 175, 'I'), (10, 2, 8, 75, 'J')
;
INSERT INTO Teams
([NBATeamId], [Name])
VALUES (1, 'Atlanta Hawks'), (2, 'Miami Heat')
;
```

PRACTICE 9

1 Identify most sold configurations for each month in 2016

2 Average time to sell a server that becomes available

SQL BOOTCAMP

Q&A

THANKS!