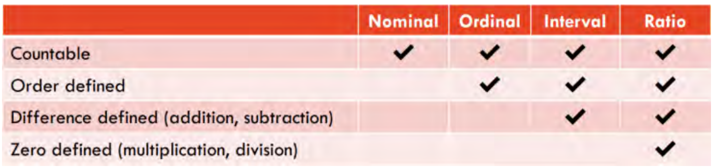
L2-L3: **Nominal Data** : Values are names; No ordering is implied; Eg jersey numbers; industry worked in; key experience you have

Ordinal Data: Values are ordered No distance is implied – Eg rank, agreement – central tendency can be measured by mode or median – **the mean cannot be defined from an ordinal set** – dispersion can be estimated by the Inter-Quartile Range (IQR) *The IQR is the difference between the first and third quartile*

Interval Data Interval scales provide information about order, and also possess equal intervals – Values encode differences – equal intervals between values – No true zero – Addition is defined – Eg **Celsius temperature** central tendency can be measured by mode, median, or mean

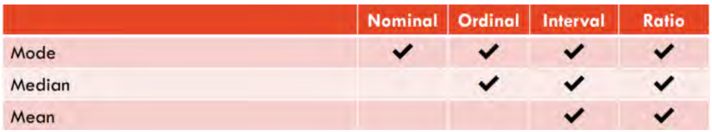
Ratio Data – Values encode differences – Zero is defined – Multiplication defined – Ratio is meaningful – Eg length, weight, income

Level of measurement



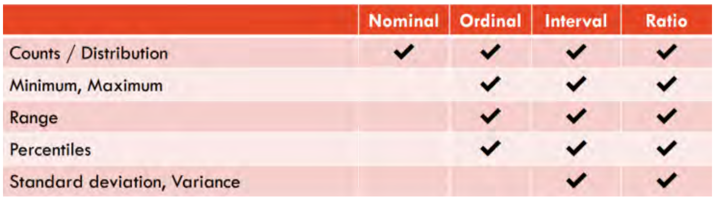
	Nominal	Ordinal	Interval	Ratio
Countable	✓	✓	✓	✓
Order defined		✓	✓	✓
Difference defined (addition, subtraction)			✓	✓
Zero defined (multiplication, division)				✓

Measure of central tendency

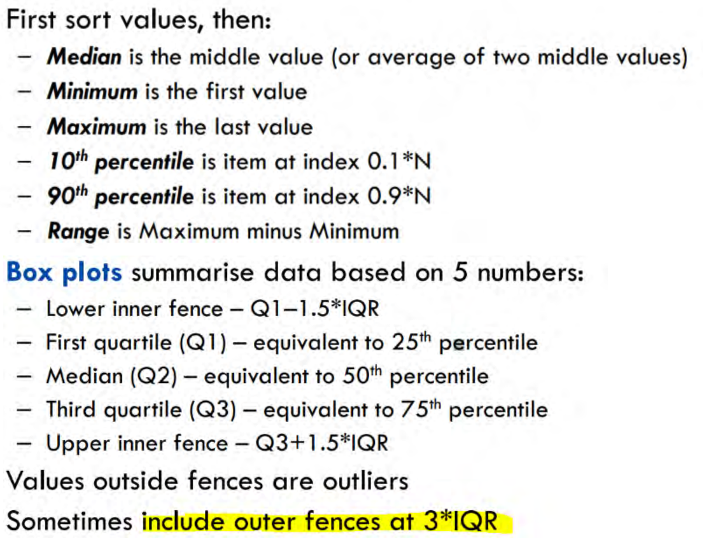


	Nominal	Ordinal	Interval	Ratio
Mode	✓	✓	✓	✓
Median		✓	✓	✓
Mean			✓	✓

Measure of Dispersion



	Nominal	Ordinal	Interval	Ratio
Counts / Distribution	✓	✓	✓	✓
Minimum, Maximum		✓	✓	✓
Range		✓	✓	✓
Percentiles		✓	✓	✓
Standard deviation, Variance			✓	✓



First sort values, then:

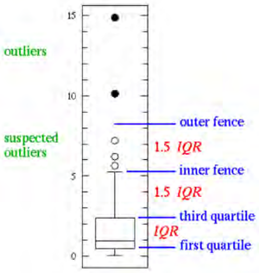
- ***Median*** is the middle value (or average of two middle values)
- ***Minimum*** is the first value
- ***Maximum*** is the last value
- ***10th percentile*** is item at index 0.1*N
- ***90th percentile*** is item at index 0.9*N
- ***Range*** is Maximum minus Minimum

Box plots summarise data based on 5 numbers:

- Lower inner fence – Q1–1.5*IQR
- First quartile (Q1) – equivalent to 25th percentile
- Median (Q2) – equivalent to 50th percentile
- Third quartile (Q3) – equivalent to 75th percentile
- Upper inner fence – Q3+1.5*IQR

Values outside fences are outliers

Sometimes include outer fences at 3*IQR



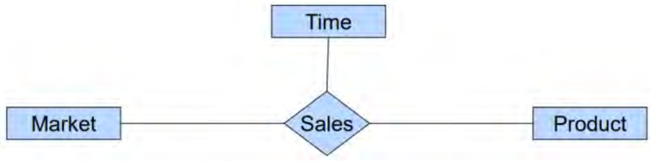
Relational data model is the most widely used model today – Main concept: relation, basically a table with rows and columns – Every relation has a schema, which describes the columns, or fields

Not all tables qualify as a relation:

– Every relation must have a **unique** name.
– **Attributes (columns)** in tables must have **unique names**.
=> The order of the columns is irrelevant.
– All tuples in a relation have the same structure; constructed from the same set of attributes
– Every attribute value is atomic (not multivalued, not composite).
– **Every row is unique** (can't have two rows with exactly the same values for all their fields)
– The order of the rows is immaterial

ETL Process: Capture/Extract - Data Cleansing - Transform – Load

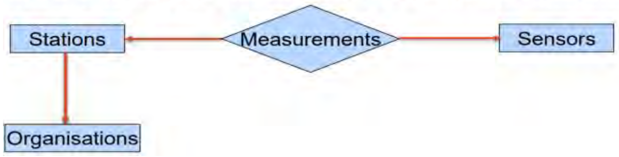
The fact and dimension relations linked to it looks like a star; – this is called a star schema



If we map this to relations

- 1 central fact table
- *n* dimension tables with foreign key relationships from the fact table *(the fact table holds the FKs referencing the dimension tables)*

Snowflake schema: A refinement of star schema where some dimensional hierarchy is **normalized** into a set of smaller dimension tables, forming a shape similar to snowflake measurements are the facts, rest describes the dimensions

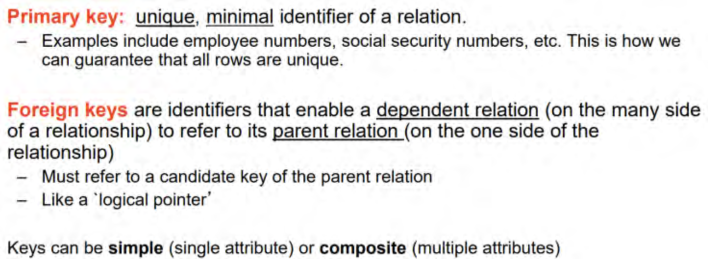


Fact constellations: Multiple fact tables share dimension tables, viewed as a collection of stars, therefore called **galaxy schema** or **fact constellation.**

DDL (Data Definition Language)

CREATE TABLE name (list_of_columns)

DML (Data Manipulation Language) for retrieval of information also called **query language** SELECT … FROM … WHERE



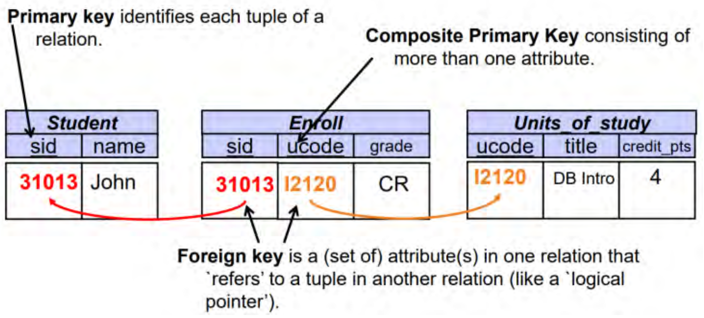
Primary key: unique, minimal identifier of a relation.

- Examples include employee numbers, social security numbers, etc. This is how we can guarantee that all rows are unique.

Foreign keys are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship)

- Must refer to a candidate key of the parent relation
- Like a `logical pointer'

Keys can be **simple** (single attribute) or **composite** (multiple attributes)



Primary key identifies each tuple of a relation.

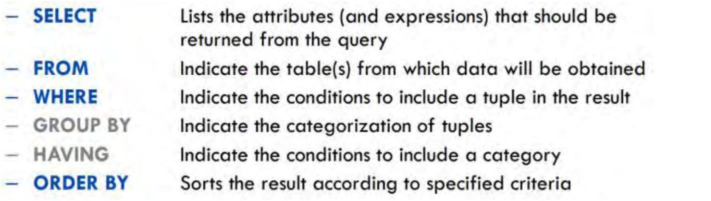
Composite Primary Key consisting of more than one attribute.

Student	
sid	name
31013	John

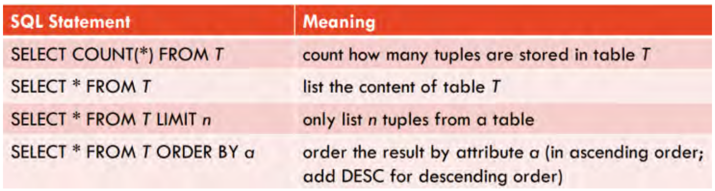
Enroll		
sid	ucode	grade
31013	I2120	CR

Units_of_study		
ucode	title	credit_pts
I2120	DB Intro	4

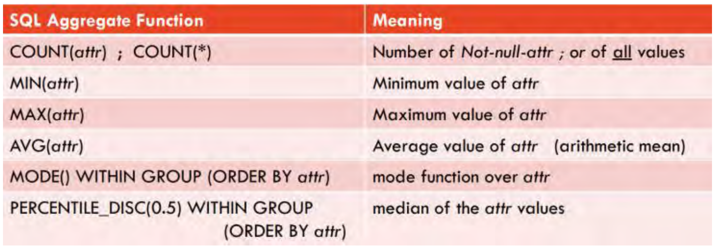
Foreign key is a (set of) attribute(s) in one relation that `refers' to a tuple in another relation (like a `logical pointer').



- **SELECT** Lists the attributes (and expressions) that should be returned from the query
- **FROM** Indicate the table(s) from which data will be obtained
- **WHERE** Indicate the conditions to include a tuple in the result
- **GROUP BY** Indicate the categorization of tuples
- **HAVING** Indicate the conditions to include a category
- **ORDER BY** Sorts the result according to specified criteria



SQL Statement	Meaning
SELECT COUNT(*) FROM *T*	count how many tuples are stored in table *T*
SELECT * FROM *T*	list the content of table *T*
SELECT * FROM *T* LIMIT *n*	only list *n* tuples from a table
SELECT * FROM *T* ORDER BY *a*	order the result by attribute *a* (in ascending order; add DESC for descending order)



SQL Aggregate Function	Meaning
COUNT(*attr*) ; COUNT(*)	Number of *Not-null-attr* ; or of all values
MIN(*attr*)	Minimum value of *attr*
MAX(*attr*)	Maximum value of *attr*
AVG(*attr*)	Average value of *attr* (arithmetic mean)
MODE() WITHIN GROUP (ORDER BY *attr*)	mode function over *attr*
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY *attr*)	median of the *attr* values

How many measurements we have done?
SELECT COUNT(*) FROM Measurement
List top five measurements ordered by date in descending order
SELECT * FROM Measurement ORDER BY date DESC limit 5;

e.g1: SELECT * FROM TelescopeConfig
WHERE (mindec BETWEEN -90 AND -50) AND (maxdec >= -45) AND (tele_array = 'H168')

e.g2 SELECT * FROM TelescopeConfig
WHERE tele_array LIKE 'H%';

EXTRACT(year FROM startDate)

TO_DATE('01-03-2012', 'DD-Mon-YYYY')

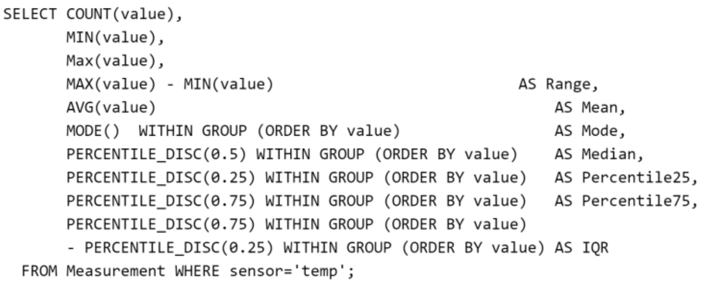
'2012-04-01' + INTERVAL '36 HOUR'

SELECT gid, band, epoch FROM Measurement WHERE intensity IS NULL
5 + null returns null

SELECT sitename, commence, organisation
FROM Station **JOIN** Organisation
ON orgcode = code; (inner join)

SELECT **uos_code** as unit_of_study, AVG(mark)
F

ROM Assessment NATURAL JOIN UnitOfStudy
WHERE credit_points = 6
GROUP BY **uos_code**
HAVING COUNT(*) > 2



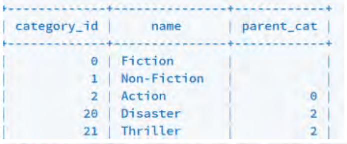
```
SELECT COUNT(value),
       MIN(value),
       Max(value),
       MAX(value) - MIN(value)                             AS Range,
       AVG(value)                                          AS Mean,
       MODE()  WITHIN GROUP (ORDER BY value)               AS Mode,
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY value)  AS Median,
       PERCENTILE_DISC(0.25) WITHIN GROUP (ORDER BY value) AS Percentile25,
       PERCENTILE_DISC(0.75) WITHIN GROUP (ORDER BY value) AS Percentile75,
       PERCENTILE_DISC(0.75) WITHIN GROUP (ORDER BY value)
       - PERCENTILE_DISC(0.25) WITHIN GROUP (ORDER BY value) AS IQR
  FROM Measurement WHERE sensor='temp';
```

In which **time period** were all the measurement done?
SELECT MIN(date), MAX(date) FROM Measurement;

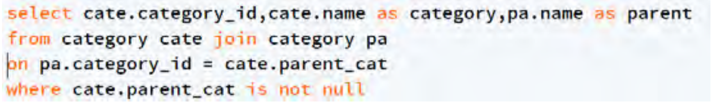
How many distinct Stations the temperature were measured
SELECT COUNT(DISTINCT station)
FROM Measurement WHERE sensor = 'temp';

How many measurements of *distinct* stations were done *per each sensor*?
SELECT sensor, COUNT(DISTINCT station)
FROM Measurement
GROUP BY sensor
ORDER BY count DESC;

self join - lists all film sub-categories and their corresponding parent categories

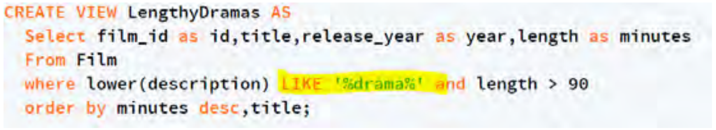


```
+-------------+--------------+------------+
| category_id |     name     | parent_cat |
+-------------+--------------+------------+
|           0 | Fiction      |            |
|           1 | Non-Fiction  |            |
|           2 | Action       |          0 |
|          20 | Disaster     |          2 |
|          21 | Thriller     |          2 |
```



```
select cate.category_id,cate.name as category,pa.name as parent
from category cate join category pa
on pa.category_id = cate.parent_cat
where cate.parent_cat is not null
```

Determines the usage of Film categories throughout our database



```
CREATE VIEW LengthyDramas AS
  Select film_id as id,title,release_year as year,length as minutes
  From Film
  where lower(description) LIKE '%drama%' and length > 90
  order by minutes desc,title;
```