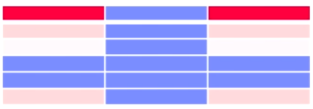
Partitioning Strategies

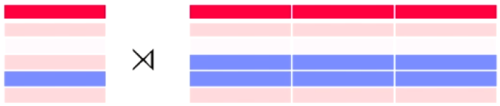
- **Horizontal Partitioning**
  - Relation partitioning into disjoint subsets

- **Vertical Partitioning**
  - Partitioning of attributes with similar access pattern

- **Hybrid Partitioning**
  - Combination of horizontal and vertical fragmentation (hierarchical partitioning)



- **Derived Horizontal Partitioning**



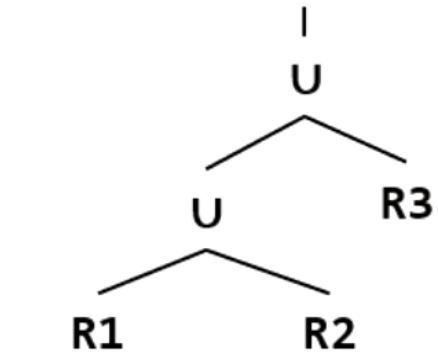
Correctness Properties

- relation R partitioned into n fragments
- completeness
  - each item from R must be included in at least one fragment
- reconstruction
  - exact reconstruction of fragments must be possible
- disjointness
  - no item must be in more than one fragment
  - $R_i \cap R_j = \emptyset$

Horizontal Partitioning

- row partitioning into n fragments
  - complete, disjoint and reconstructable
  - schema of fragments is equivalent to schema of base relation
- partitioning
  - split table by n selection predicates on attributes
    - e.g. split by last name
  - beware of attribute domain and skew

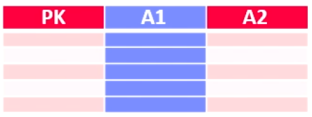
- reconstruction
  - union of all fragments
  - bag semantics but no duplicates across partitions



$$R = \bigcup_{1 \le i \le n} R_i$$

Vertical Fragmentation

- **Column Partitioning into n Fragments Ri**
  - **Complete**, **reconstructable**, but not disjoint (**primary key** for reconstruction via join)
  - Completeness: each attribute must be included in at least one fragment



- **Partitioning**
  - Partitioning via **projection**
  - Redundancy of primary key

$$R_i = \pi_{PK,A_i}(R) \quad (1 \le i \le n)$$

- **Reconstruction**
  - **Natural join** over primary key

$$R = R_1 \bowtie R_i \bowtie R_n \quad (1 \le i \le n)$$

- **Hybrid horizontal/vertical partitioning**

$R = R_1 \bowtie R_i \bowtie R_n$ w/ $R_i = \cup R_{ij}$
→ $R = \cup R_j$ w/ $R_j = R_{1j} \bowtie R_{ij} \bowtie R_{nj}$

Derived Horizontal Fragmentation

- **Row Partitioning R into n fragements R_i, with partitioning predicate on S**
  - Potentially complete (not guaranteed), **restructable**, **disjoint**
  - Foreign key / primary key relationship determines correctness

- **Partitioning**
  - **Selection** on independent relation S
  - **Semi-join** with dependent relation R to select partition R_i

$$R_i = R \ltimes S_i = R \ltimes \sigma_{P_i}(S) = \pi_{R.*}\left(R \bowtie \sigma_{P_i}(S)\right)$$

- **Reconstruction**
  - Equivalent to horizontal partitioning
  - **Union** of all fragments

$$R = \bigcup_{1 \le i \le n} R_i$$

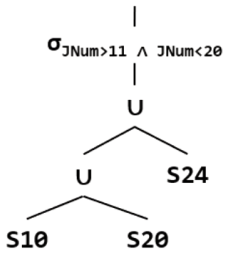
Exploiting Partitioning in Postgre

- **Partitioning and query rewriting**
  - **#1** Manual partitioning and rewriting
  - **#2** Automatic rewriting (spec. partitioning)
  - **#3** Automatic partitioning and rewriting
- **Example PostgreSQL (#2)**

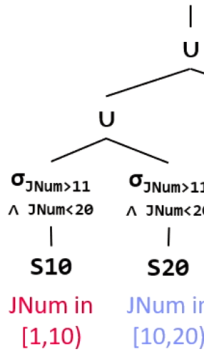
```
CREATE TABLE Squad(
   JNum INT PRIMARY KEY,
   Pos CHAR(2) NOT NULL,
   Name VARCHAR(256)
) PARTITION BY RANGE(JNum);

CREATE TABLE Squad10 PARTITION OF Squad
   FOR VALUES FROM (1) TO (10);
CREATE TABLE Squad20 PARTITION OF Squad
   FOR VALUES FROM (10) TO (20);
CREATE TABLE Squad24 PARTITION OF Squad
   FOR VALUES FROM (20) TO (24);
```

J#	Pos	Name
1	GK	Manuel Neuer
12	GK	Ron-Robert Zieler
22	GK	Roman Weidenfeller
2	DF	Kevin Großkreutz
4	DF	Benedikt Höwedes
5	DF	Mats Hummels
15	DF	Erik Durm
16	DF	Philipp Lahm
17	DF	Per Mertesacker
20	DF	Jérôme Boateng
3	MF	Matthias Ginter
6	MF	Sami Khedira
7	MF	Bastian Schweinsteiger
8	MF	Mesut Özil
9	MF	André Schürrle
13	MF	Thomas Müller
14	MF	Julian Draxler
18	MF	Toni Kroos
19	MF	Mario Götze
21	MF	Marco Reus
23	MF	Christoph Kramer
10	FW	Lukas Podolski
11	FW	Miroslav Klose

- **Example, cont.**

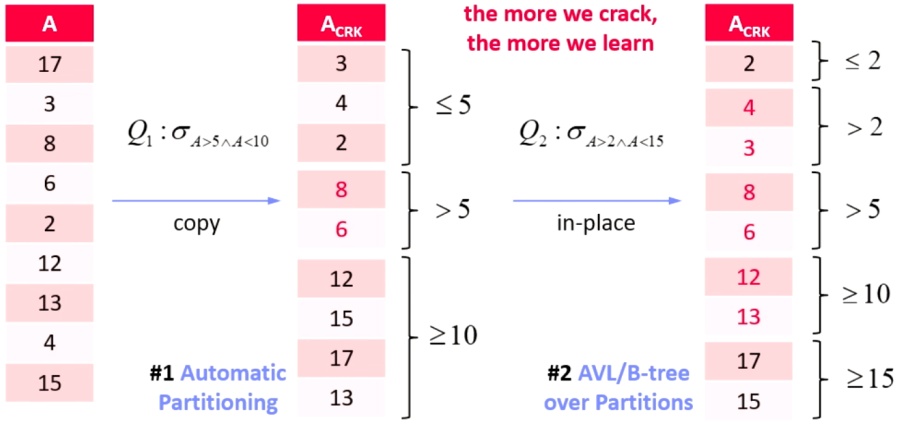



```
SELECT * FROM
   WHERE JNum
```



Database Cracking

- database layout adapts to requested queries and their range predicates
  - creates [[Index Structures]] to identify qualifying partitions
  - inside partition its elements are unordered
- workload creates hybrid between partitioning and indexing



[[Background Storage System]] [[Database Performance Tuning]]