

Relational Algebra

Basic Operation

Select σ

Project Π

Union \cup

Set different $-$

Product \times

Rename ρ

Division $/$

Example

- *Select Relation $a = b$ and $d > 5 = \sigma_{a=b \wedge d > 5}(R)$*
- *$\Pi_{f_name, l_name}(\sigma_{dno=4 \wedge salary > 25000}(Employee))$*
- *Rename E . sid to $C = \rho(C(sid \rightarrow identity), E)$*
- *Condition Join $= R \bowtie_c S = \sigma_c(R \times S)$*

SQL

Basic SQL Query

```
1 SELECT [DISTINCT] target-list
2 FROM relation-list
3 WHERE qualification
```

- Equal Join with tables

```
1 SELECT S.sname
2 FROM Sailors S, Reserves R, Boat B
3 WHERE S.sid = R.sid AND R.bid = B.bid
```

- Regex in SQL

```
1 SELECT S.age, age1=S.age-5, 2*S.age AS age2
2 FROM Sailors S
3 WHERE S.sname LIKE 'B_%B'
```

`LIKE` for string matching, " _ " means any characters, " % " means arbitrary

- **Set-manipulation construction**

- UNION \cup
INTERSECTION \cap
EXCEPT $-$

```
1 SELECT S.sid
2 FROM Sailors S, Boats B, Reserves R
3 WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
4 INTERSECT
5 SELECT S.sid
6 FROM Sailors S, Boats B, Reserves R
7 WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

```
1 SELECT R.sid
2 FROM Boats B, Reserves R
3 WHERE R.bid=B.bid AND B.color='red'
4 EXCEPT
5 SELECT R.sid
6 FROM Boats B, Reserves R
7 WHERE R.bid=B.bid AND B.color='green'
```

- **Correlated Nested Query**

```
1 SELECT S.sname
2 FROM Sailors S
3 WHERE EXISTS(SELECT * FROM Reserves R WHERE R.bid=103 AND S.sid=R.sid)
```

- EXISTS to test for nonempty
- IN operator specified **multiple values** in WHERE clause

```
1 SELECT B.bname
2 FROM Boats B
3 WHERE B.color IN ('red', 'blue', 'green')
```

- **Set-comparison operators**

- ops ANY Or ops ALL

```
1 SELECT *
2 FROM Sailors S
3 WHERE S.rating > ANY (SELECT S2.rating FROM Sailors S2 WHERE
                        S2.sname='Horatio')
```

- **Division in SQL**

```

1 SELECT S.sname
2 FROM Sailors S
3 WHERE NOT EXISTS
4     ((SELECT B.bid
5        FROM Boats B)
6      EXCEPT
7      (SELECT R.bid
8        FROM Reserves R
9        WHERE R.sid=S.sid))

```

SQL Aggregate Operators	Description
<code>COUNT(A)</code>	Number of values in A column
<code>SUM(A)</code>	Sum of all values in A column
<code>AVG(A)</code>	Average of all values on A column
<code>MAX(A)</code>	Maximum value in the A column
<code>MIN(A)</code>	Minimum value in the A column

- Correct way to use the above operators

```

1 SELECT S.name
2 FROM Sailors S
3 WHERE S.age > (SELECT MAX (S2.age)
4                FROM Sailors S2
5                WHERE S2.rating = 10)

```

- `GROUP BY` similar as for-loop

```

1 SELECT S.rating, MIN(S.age)
2 FROM Sailors S
3 GROUP BY S.rating
4
5 For i = 1, 2, ..., 10:
6     SELECT MIN(S.age)
7     FROM Sailors S
8     WHERE S.rating = i

```

- Columns appeared in `GROUP BY` should use `HAVING`
- `CREATE VIEW` for creating virtual table based on result SQL statement

```

1 CREATE VIEW Temp AS
2   SELECT S.rating, AVG (S.age) AS avgage
3   FROM Sailors S
4   GROUP BY S.rating
5
6 DROP VIEW temp

```

Schema Refinement

Functional Dependencies

```

1 Let X and Y be nonempty sets of attributes in R
2 An instance r of R satisfies the FD X->Y if
3 If t1.X = t2.X, then t1.Y = t2.Y

```

Trivially Preserved

```

1 If any two row never have the same value for a in a->b
2 Then a->b is trivially preserved

```

Trivially Dependency

```

1 If right hand side of arrow is subset of that on left hand side a->b
2 Then a->b is a trivial dependency

```

Closure of set

- Given a set F , the set of all FDs implied is called the closure of F , denoted as F^+

Armstrong's Axioms and additional rules

- Reflexivity:** *if $Y \subseteq X$, then $X \rightarrow Y$*
- Augmentation:** *if $X \rightarrow Y$, then $XZ \rightarrow YZ$*
- Transitivity:** *if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$*
- Union:** *if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$*
- Decomposition:** *if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$*

Boyce-Codd Normal Form (BCNF)

R — a relation schema

F — set of functional dependencies on R

R is in BCNF if for any $X \rightarrow A$ in F ,

- $X \rightarrow A$ is a trivial functional dependency, i. e. $A \subseteq X$

OR

- *X is a superkey for R*

Third Normal Form

- If R is in BCNF, then it is also in 3NF since $3NF \subset BCNF$

R — a relation schema

F — set of functional dependencies on R

R is in 3NF if for any $X \rightarrow A$ in *F*,

- *X → A is a trivial functional dependency, i.e. $A \subseteq X$*

OR

- *X is a superkey for R*

OR

- *A is part of some key for R*

Decomposition

1. Remove the rule $X \rightarrow A$ that violates the condition from relation schema R
2. Create a new relational schema XA

Lossless Join Decomposition

R — a relation schema

F — set of functional dependencies on R

The decomposition of R into relations with attribute sets R_1, R_2 is lossless — join iff

$$(R_1 \cap R_2) \rightarrow R_1 \in F^+$$

OR

$$(R_1 \cap R_2) \rightarrow R_2 \in F^+$$

$R_1 \cap R_2$ is a superkey for R_1 or R_2

Dependency preserved

$$(F_1 \cup F_2)^+ = F^+$$

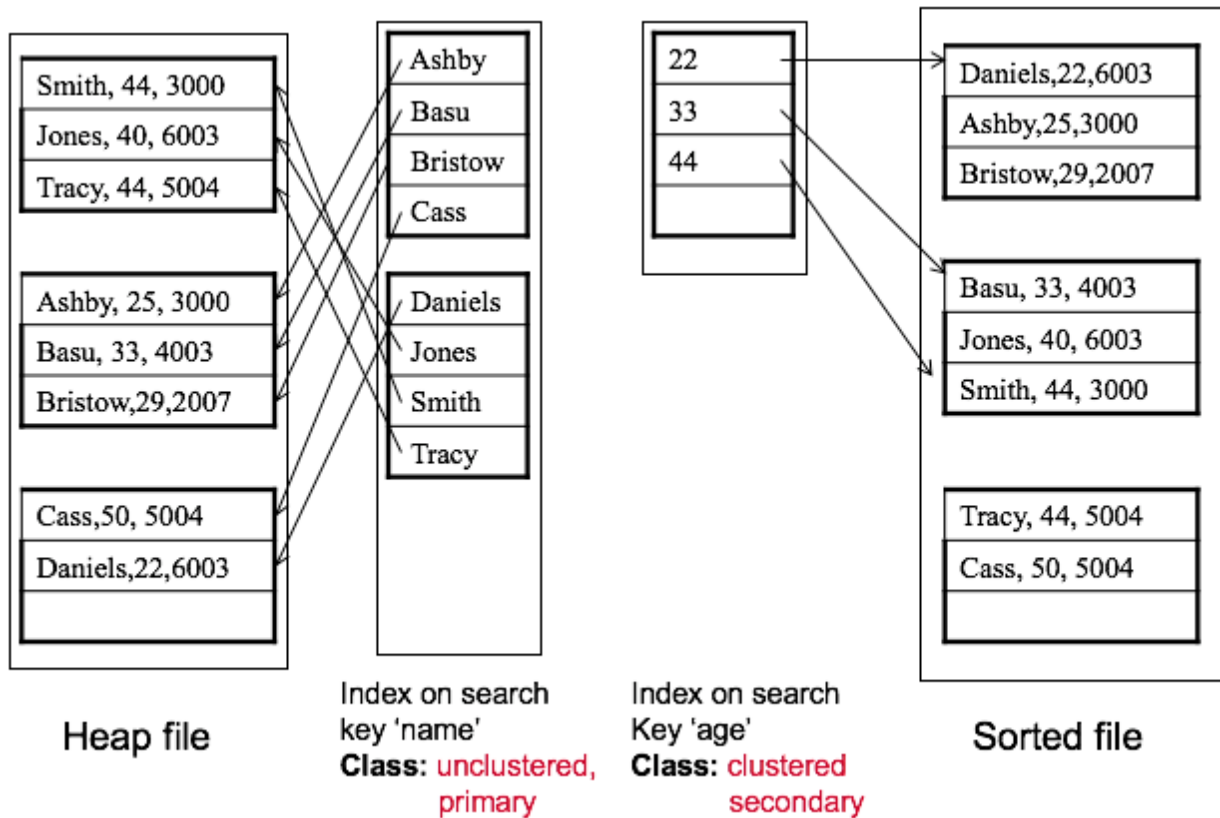
- Possible to obtain lossless-join decomposition into collection of BCNF relation schemas
non — $BCNF \rightarrow BCNF$ but **NOT** guaranteed dependency-preserving
- Always exists a dependency-preserving, lossless-join decomposition into collection of 3NF relation schemas
non — $3NF \rightarrow 3NF$

Canonical Cover

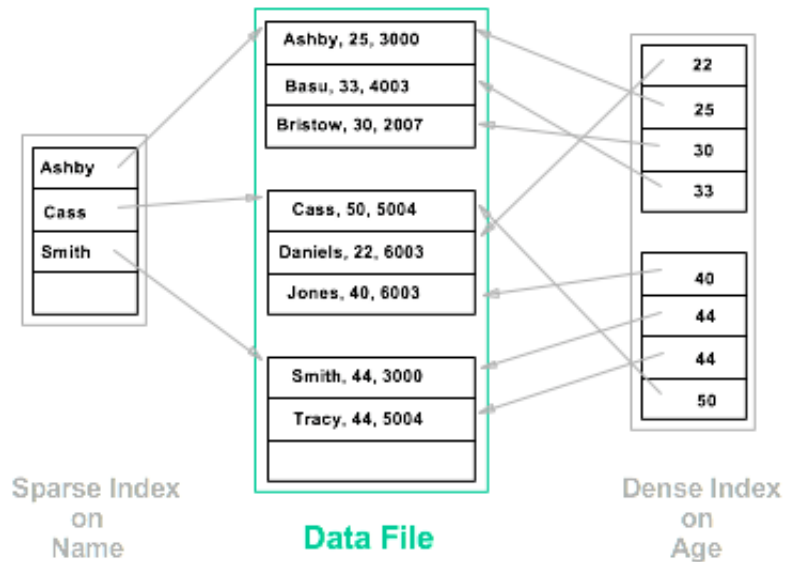
- A **minimal and equivalent** set of functional dependency

Storage and Index

- Index on file speeds up selections on **search key fields**
- Search key can be any subset of fields of relation



Primary index	Contains primary key in search key
Secondary index	Does not contain primary key in search key
Clustered index	Order of data records close to order of data entries
Unclustered index	
Dense index	At least one data entry per search key value
Sparse index	Every sparse index is clustered



- **Unclustered** must be also in **dense**
 - Primary: each data entry k^* points to **single record** that contains k
 - Secondary: each data entry k^* points to **all records** that contains k
- **Clustered** must be also in **sparse**
 - Sort both data file and index file on search key
 - Each data entry k^* points to the **first record** that contains k
 - Overflow pages may be needed for inserting, so the order is **closed to** sorted

Composite Search Keys

Equality query	Every field value is equal to a constant value
Range query	Some field value is not a constant

Tree-Structured Indexing

- Example of B+ tree

