GROUP BY and HAVING

So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.

Consider: *Find the age of the youngest student for <u>each</u> group.*

- In general, we don't know how many groups exist
- Suppose we know that group values go from 110 to 119, we can write 10 similar queries. But when another group is added, a new query should be created.

Group By and *Having* clauses allow us to solve problems like this in only one SQL query. General syntax is:

```
SELECT [DISTINCT] target-list
FROM   relation-list
WHERE  qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

The *target-list* contains

- <u>attribute names</u> (the <u>attribute names</u> must be a subset of *grouping-list*);
- terms with aggregate operations (e.g., MIN (*S.age*)).

Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Group By / *Having* conceptual evaluation:

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary'* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a <u>*single value per group*</u>!
    - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Sample: *Find the age of the youngest student with age ≥ 20 for each group with at least 2 such students*

```
SELECT  S.gr,  MIN (S.age)
FROM  Students S
WHERE  S.age >= 20
GROUP BY  S.gr
HAVING  COUNT (*) > 1
```

- Only S.gr and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other attributes `*unnecessary*'.
- 2nd column of result is unnamed. (Use AS to name it.)

Sample: *Find the number of enrolled students and the grade average for each course with 6 credits*

```
SELECT  C.cid,  COUNT (*) AS scount, AVG(grade)
FROM  Students S, Enrolled E, Courses C
```

```
WHERE  S.sid=E.sid AND E.cid=C.cid AND C.credits=6
GROUP BY  C.cid
```

Sorting the result of a query

ORDER BY *column* [ASC | DESC] [, ...]

```
SELECT cname, sname, grade
FROM Courses C
      INNER JOIN Enrolled E ON C.cid = E.cid
      INNER JOIN Students S ON E.sid = S.sid
ORDER BY cname, grade DESC , sname
```

Can order by any column in SELECT list, including expressions or aggregates:

```
SELECT gr, Count(*) as StudNo
FROM Students C
GROUP BY gr
ORDER BY StudNo
```

Course 3. Schema Refinement

Good designs / bad designs

Data represented by schemas generally have application-dependent constraints relating to attribute values.

Example: Consider the following *MovieList* relation:

Title	Director	Cinema	Phone	Time
The Hobbit	Jackson	Cinema City	441111	11:30
The Lord of the Rings 3	Jackson	Cinema City	441111	14:30
Adventures of Tintin	Spielberg	Odeon	442222	11:00
The Lord of the Rings 3	Jackson	Odeon	442222	14:00
War Horse	Spielberg	Odeon	442222	16:30

Figure 3.1 *MovieList* relation instance

Data stored by this relation respect the following constraints:

- Each movie has one director
- Each cinema has one phone number
- Each cinema screens one movie at a time

Common problems if a design is bad:

- **Insertion anomaly**: We can't store information about a new movie if the screening place and time are not known
- **Deletion anomaly**: If we delete all movies directed by Peter Jackson, we lose information about *Cinema City* cinema

- **Update anomaly**: If the phone number of a cinema changes, we have to be careful of inconsistent updates

Usually, we can refine a bad schema by *decomposing* it into multiple "good" ones.

Movies

Title	Director
The Hobbit	Jackson
The Lord of the Rings 3	Jackson
Adventures of Tintin	Spielberg
War Horse	Spielberg

Cinema

Cinema	Phone
Cinema City	441111
Odeon	442222

Screens

Cinema	Time	Title
Cinema City	11:30	The Hobbit
Cinema City	14:30	The Lord of the Rings 3
Odeon	11:00	Adventures of Tintin
Odeon	14:00	The Lord of the Rings 3
Odeon	16:30	War Horse

Figure 3.2 Decomposition of *MovieList* relation

Refined schema allows:

– insertions of new movies without knowing their screening details
– deletions of movies without losing information about cinemas
– a single record to be updated to change a cinema's phone number

There are two main questions:

- How to determine whether a schema design is "*good*" or "*bad*"?
- How to transform a bad design into a *good* one?

The theory of *functional dependencies* provides a systematic approach to address these questions. This theory was introduced by E.F. Codd in: "*A relational model for large shared data banks*", Com. of the ACM, 13(6), 1970, pp.377-387.

Functional dependencies

Functional dependencies (FDs) are constraints on schemas that specify that the values for a certain set of attributes determine unique values for another set of attributes

Let α and β denote subsets of attributes of a relational schema R. We use:

$$\alpha \rightarrow \beta$$

to denote that α functionally determines β (or β functionally depends on α)

In our previous example (*MovieList* relation) we can identify the following functional dependencies:

1. Title $\rightarrow$ Director

2. Cinema → Phone

3. Cinema, Time → Title

Definition. The functional dependency $\alpha \rightarrow \beta$ holds on R if and only if for any relation instance of R, whenever two tuples t_1 and t_2 agree on the attributes α, they also agree on the attributes β.

That is,

$$\pi_\alpha(t_1) = \pi_\alpha(t_2) \quad \Rightarrow \quad \pi_\beta(t_1) = \pi_\beta(t_2)$$

Note: $\pi_\alpha(t)$ denote the projection of attributes α of tuple t

Let *r* be a relation instance of relation schema *R*

We are saying that *r* **satisfies FD** $\alpha \rightarrow \beta$ if for every pair of tuples t_1 and t_2 in *r* such that $\pi_\alpha(t_1) = \pi_\alpha(t_2)$, it is also true that $\pi_\beta(t_1) = \pi_\beta(t_2)$. Thus, a **FD** *f* **holds on** *R* if and only if for any relation instance *r* of *R*, *r* satisfies *f*

r is said to **violate** a FD *f* if *r* does not satisfy *f*. *r* is said to be a **legal instance of R** if *r* satisfies all FDs that hold on *R*.

A FD $\alpha \rightarrow \beta$ is a **trivial FD** if $\alpha \supseteq \beta$; otherwise it is a **non-trivial FD**

Example. Relation ***Movie***(Title, Director, Composer). Let *r* be a legal relation instance of ***Movie*** as shown:

Title	Director	Composer
Schindler's List	Spielberg	Williams
Saving Private Ryan	Spielberg	Williams
North by Northwest	Hitchcock	Herrmann
Angela's Ashes	Parker	Williams
Vertigo	Hitchcock	Herrmann

Figure 3.3. ***Movie*** relation instance

The functional dependency *composer* → *director* does not hold on ***Movie***. At the same time, *r* satisfies the FD *director* → *composer*, but we cannot conclude that *director*→*composer* holds on ***Movie***!

Conclusion: based on legal instances of R we can tell which FDs do not hold on R, but we can't deduce which non-trivial FDs hold.

Implication Problem: Given a set of functional dependencies *F* (that hold on *R*) and a functional dependency *f*, does *f* also hold on *R*? *F* **logically implies (or implies)** *f*, denoted by $F \Rightarrow f$, if every relation instance *r* of *R* that satisfies the FDs *F* also satisfies the FD *f*

Example: In ***MovieList***, we have the following predefined set of functional dependencies:

F = { Title → Director

Cinema → Phone

Cinema, Time $\rightarrow$ Title }

Does *Cinema, Time* $\rightarrow$ *Director* or *Time* $\rightarrow$ *Director* also hold?

Let F & G denote sets of functional dependencies, and f denote a functional dependency. More generally, $F \Rightarrow G$ if $F \Rightarrow g$ for each $g \in G$.

The **closure of F** (denoted by F^+) is the set of all functional dependencies implied by F; that is,

$$F^+ = \{f \mid F \Rightarrow f\}$$

\Two sets of functional dependencies, F and G, are **equivalent** (denoted by $F \equiv G$) if $F^+ = G^+$ (i.e., $F \Rightarrow G$ and $G \Rightarrow F$)

Axioms for Functional Dependencies

= a collection of formal rules used to derive a functional dependency from a set of functional dependencies

Armstrong's Axioms: Let $\alpha, \beta, \gamma \subseteq R$

Reflexivity: If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$

Augmentation: If $\alpha \rightarrow \beta$, then $\alpha\gamma \rightarrow \beta\gamma$

Transitivity: If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

Armstrong's Axioms are both *sound* and *complete*

Sound: Any derived FD is implied by F

Complete: All FDs in F^+ can be derived

Problem: Consider R(A, B, C, D, E) with 3 functional dependencies:

$$F = \{A \rightarrow C;\ B \rightarrow C;\ CD \rightarrow E\}.$$

Show that F $\Rightarrow$ AD $\rightarrow$ E

Solution:

1. A $\rightarrow$ C (given)
2. AD $\rightarrow$ CD (augmentation with (1))
3. CD $\rightarrow$ E (given)
4. AD $\rightarrow$ E (transitivity with (2) and (3))

Additional Inference Rules

Union: If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then $\alpha \rightarrow \beta\gamma$

Decomposition: If $\alpha \rightarrow \beta$, then $\alpha \rightarrow \beta'$ for any $\beta' \subseteq \beta$

Superkeys, Keys & Prime Attributes

A set of attributes α is a superkey of schema R (with FDs F) if $F \Rightarrow \alpha \rightarrow R$.

A set of attributes is a key of schema R if

(1) α is a superkey, and

(2) no proper subset of α is a superkey

(i.e., for each $\beta \subset \alpha$, $\beta \rightarrow R \notin F^+$)

An attribute $A \in R$ is a prime attribute if A is contained in some key of R; otherwise, it is a nonprime attribute.

Example: Consider again ***MovieList*** (Title, Director, Cinema, Phone, Time) with functional dependencies set:

(1) Cinema, Time $\rightarrow$ Title

(2) Cinema $\rightarrow$ Phone

(3) Title $\rightarrow$ Director

{*Cinema, Time*} is the only key of ***MovieList***.

Cinema and *Time* are the only prime attributes in ***MovieList***.

Any superset of {*Cinema; Time*} in R is a superkey of ***MovieList***.

Attribute Closure

Computing F^+ for a set of FDs F is not efficient as the size of F^+ could be exponentially large!

More efficient to compute the closure of a set of attributes

Let $\alpha \subseteq R$ and F be a set of FDs that hold on R. The closure of α (with respect to F), denoted by α^+, is the set of attributes that are functionally determined by α with respect to F; i.e.,

$$\alpha^+ = \{A \in R \mid F \Rightarrow \alpha \rightarrow A\}$$

Note that $F \Rightarrow \alpha \rightarrow \beta$ if and only if $\beta \subseteq \alpha^+$(w.r.t. F)

Algorithm to compute attribute closure:

Input: α, F

Output: α^+ (w.r.t. F)

Compute a sequence of sets of attrs α_0, α_1,... α_k, α_{k+1} as follows:

$\alpha_0 = \alpha$

$\alpha_{i+1} = \alpha_i \cup \gamma$ such that there is some FD

$\beta \rightarrow \gamma \in F$ and $\beta \subseteq \alpha_i$

Terminate the computation once $\alpha_{k+1} = \alpha_k$ for some *k* Return α_k

Problem: Given $F = \{A \rightarrow C; B \rightarrow C; CD \rightarrow E\}$, show that $F \Rightarrow AD \rightarrow E$.

Solution

i	α_i	*FD used*
0	AD	given input
1	ACD	A→C
2	ACDE	CD → E
3	ACDE	none

So $AD^+ = ACDE$. Since $E \in AD^+$, therefore $F \Rightarrow AD \rightarrow E$

Schema Decompositions

The **decomposition of schema R** is a set of schemas $\{R_1, R_2, ..., R_n\}$ such that each $R_i \subseteq R$ and $R = \cup R_i$. If *r* is a relation of R, then *r* is decomposed into $\{r_1, r_2,..., r_n\}$, where each $r_i = \pi_{R_i}(r)$

Example:

{ (Cinema, Time, Title),
(Title, Director),
(Cinema, Phone)}

is a decomposition of: ***MovieList***(Title, Director, Cinema, Phone, Time)

Properties of schema decomposition:

1. Decomposition must preserve information
   - Data in original relation ≡ Data in decomposed relations
   - Crucial for correctness!
2. Decomposition should preserve FDs
   - Functional dependencies in original schema ≡ functional dependencies in decomposed schemas
   - Facilitates checking of functional dependency violations

Lossless - Join Decomposition

It is important that a decomposition preserves information; i.e., we can reconstruct *r* from joining its projections $\{r_1, r_2, ... , r_n\}$. Note that if $\{R_1, R_2, ... , R_n\}$ is a decomposition of R, then for any relation *r* of R, it is always true that

$$r \subseteq \pi_{R1}(r) \otimes \pi_{R2}(r) \otimes ... \otimes \pi_{Rn}(r)$$

A decomposition of R (with functional dependencies set *F*) into $\{R_1,R_2,...,R_n\}$ is a lossless-join decomposition with respect to *F* if

$$\pi_{R1}(r) \otimes \pi_{R2}(r) \otimes ... \otimes \pi_{Rn}(r) = r$$

for every relation r of R that satisfies *F*.

Example. Consider the decomposition of R(A,B,C) into $\{R_1(AC), R_2(BC)\}$

r

A	B	C
a_1	b_1	c
a_1	b_2	c
a_2	b_1	c

→

r_1

A	C
a_1	c
a_2	c

r_2

B	C
b_1	c
b_2	c

→

$r_1 \otimes r_2$

A	B	C
a_1	b_1	c
a_1	b_2	c
a_2	b_1	c
a_2	b_2	c

Since $r \subset r_1 \otimes r_2$, the above decomposition is not lossless-join A decomposition that is not lossless-join is called a lossy decomposition

How to determine if $\{R_1, R_2\}$ is a lossless-join decomposition of R?

Theorem: The decomposition of R (with FDs F) into $\{R_1, R_2\}$ is lossless with respect to F if and only if:

$F \Rightarrow R_1 \cap R_2 \rightarrow R_1$ or $F \Rightarrow R_1 \cap R_2 \rightarrow R_2$

How to decompose R into $\{R_1, R_2\}$ such that it is a lossless-join decomposition?

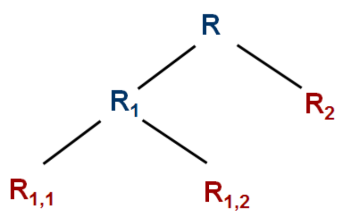
Corollary: If $\alpha \rightarrow \beta$ holds on R and $\alpha \cap \beta = \varnothing$, then the decomposition of R into $\{R- \beta , \alpha\beta\}$ is a lossless-join decomposition

Example. Consider R(A,B,C) with FDs F = { $A \rightarrow B$}

The decomposition {AB, AC} has a lossless join since $AB \cap AC = A$ and $A \rightarrow AB$

The decomposition {AB, BC} is not lossless join w.r.t. F since $AB \cap BC = B$ and neither $B \rightarrow AB$ nor $B \rightarrow BC$ holds on R.

Theorem: If $\{R_1, R_2\}$ is a lossless-join decomposition of R, and if $\{R_{1,1}, R_{1,2}\}$ is a lossless-join decomposition of R_1, then $\{R_{1,1}, R_{1,2} , R_2\}$ is a lossless-join decomposition of R :



MovieList

Title	Director	Cinema	Phone	Time
The Hobbit	Jackson	Cinema City	441111	11:30
The Lord of the Rings 3	Jackson	Cinema City	441111	14:30
Adventures of Tintin	Spielberg	Odeon	442222	11:30
War Horse	Spielberg	Odeon	442222	14:00
The Lord of the Rings 3	Jackson	Odeon	442222	16:30

Movie

Title	Director
The Hobbit	Jackson
The Lord of the Rings 3	Jackson
Adventures of Tintin	Spielberg
War Horse	Spielberg

Cinema-Screens

Cinema	Phone	Time	Title
Cine. City	441111	11:30	The Hobbit
Cine. City	441111	14:30	The Lord of the Rings 3
Odeon	442222	11:30	Adventures of Tintin
Odeon	442222	14:00	War Horse
Odeon	442222	16:30	The Lord of the Rings 3

Figure 3.4 First step of decomposing ***MovieList*** relation, based on

Title $\rightarrow$ *Director* functional dependency

Movie

Title	Director
The Hobbit	Jackson
The Lord of the Rings 3	Jackson
Adventures of Tintin	Spielberg
War Horse	Spielberg

Cinema-Screens

Cinema	Phone	Time	Title
Cine. City	441111	11:30	The Hobbit
Cine. City	441111	14:30	The Lord of the Rings 3
Odeon	442222	11:30	Adventures of Tintin
Odeon	442222	14:00	War Horse
Odeon	442222	16:30	The Lord of the Rings 3

Cinema

Cinema	Phone
Cine. City	441111
Odeon	442222

Screens

Cinema	Time	Title
Cine. City	11:30	The Hobbit
Cine. City	14:30	Saving Private Ryan
Odeon	11:30	Adventures of Tintin
Odeon	14:00	War Horse
Odeon	16:30	Saving Private Ryan

Figure 3.5 Last step of decomposing ***MovieList*** relation, based on *Cinema* → *Phone* functional dependency

Preserve Functional Dependencies

The projection of F on α (denote by F_α) is the set of FDs in F^+ that involves only attributes in α; i.e., $F_\alpha = \{ \beta \rightarrow \gamma \in F^+ \mid \beta\gamma \subseteq \alpha \}$

Computing FD Projections:

```
Input: α, F
Output: Fα
result = ∅;
for each  β ⊆ α do
   T =  β⁺ (w.r.t. F)
   result = result ∪ {β→T ∩ α}
return result
```

If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold.

Definition. The decomposition $\{R_1, R_2, ..., R_n\}$ of R is dependency preserving if

$(F_{R1} \cup F_{R2} \cup ... \cup F_{Rn})$ and F are equivalent, i.e.:

$(F_{R1} \cup F_{R2} \cup ... \cup F_{Rn}) \Rightarrow F$ and $F \Rightarrow (F_{R1} \cup F_{R2} \cup ... \cup F_{Rn})$