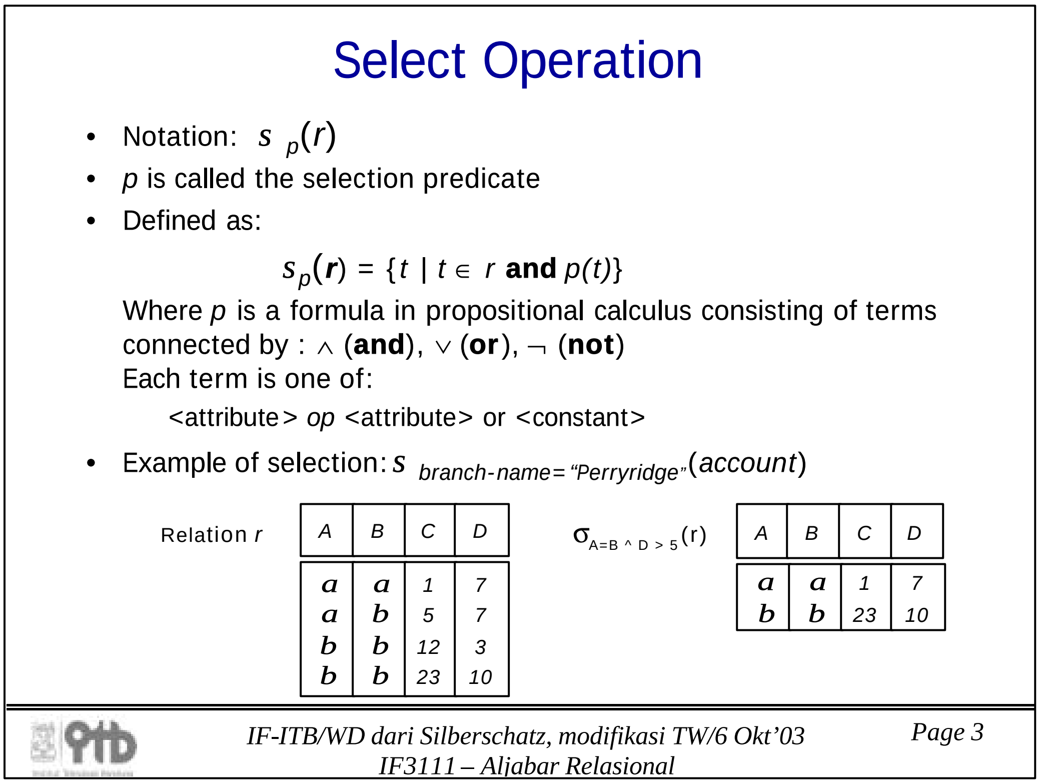
IF3111 - Operasi Aljabar Relasional

Wikan Danar
Departemen Teknik Informatika
Institut Teknologi Bandung

Relational Algebra

- Procedural language
- Six basic operators
  - select
  - project
  - union
  - set difference
  - cartesian product
  - rename
- The operators take one or more relations as inputs and give a new relation as a result.



Select Operation

- Notation: $s_{p}(r)$
- p is called the selection predicate
- Defined as:

$$s_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of terms connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each term is one of:

<attribute> *op* <attribute> or <constant>

- Example of selection: $s_{branch\text{-}name=\text{“Perryridge”}}(account)$

Relation *r*

A	B	C	D
a	a	1	7
a	b	5	7
b	b	12	3
b	b	23	10

$\sigma_{A=B\ \wedge\ D\ >\ 5}(r)$

A	B	C	D
a	a	1	7
b	b	23	10

op is one of: =, ≠, >, ≥. <. ≤

Project Operation

- Notation:

  $$\Pi_{A1,\ A2,\ \ldots,\ Ak}\ (r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- E.g. To eliminate the *branch-name* attribute of *account*

  $$\Pi_{account\text{-}number,\ balance}\ (account)$$

Relation *r*

A	B	C
a	10	1
a	20	1
b	30	1
b	40	2

$\Pi_{A,C}\ (r)$

A	C
a	1
a	1
b	1
b	2

=

A	C
a	1
b	1
b	2

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same arity* (same number of attributes)
  2. The attribute domains must be *compatible* (e.g., 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- E.g. to find all customers with either an account or a loan
  $\Pi_{customer\text{-}name}$ (*depositor*) $\cup\ \Pi_{customer\text{-}name}$ (*borrower)*

r

A	B
a	1
a	2
b	1

s

A	B
a	2
b	3

$r \cup s$:

A	B
a	1
a	2
b	1
b	3

Set Difference Operation

- Notation *r* – *s*
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between *compatible* relations.
  - *r* and *s* must have the *same arity*
  - attribute domains of *r* and *s* must be compatible

r

A	B
a	1
a	2
b	1

s

A	B
a	2
b	3

r – *s*:

A	B
a	1
b	1

Cartesian-Product Operation

- Notation *r* x *s*
- Defined as:

  r x *s* = {*t q* | *t* ∈ *r* **and** *q* ∈ *s*}
- Assume that attributes of r(R) and s(S) are disjoint. (That is, *R* ∩ *S* = Æ).
- If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

r

A	B
a	1
b	2

s

C	D	E
a	10	a
b	10	a
b	20	b
g	10	b

r x *s*

A	B	C	D	E
a	1	**a**	10	a
a	1	**b**	10	a
a	1	**b**	20	b
a	1	**g**	10	b
b	2	**a**	10	a
b	2	**b**	10	a
b	2	**b**	20	b
b	2	**g**	10	b

Rename Operation

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.

Example:

$$r_{X}(E)$$

returns the expression E under the name X

If a relational-algebra expression E has arity n, then

$$r_{X\,(A1,\,A2,\,\ldots,\,An)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A1, A2, \ldots., An$.

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r\ x\ s)$

r x s

A	B	C	D	E
a	*1*	***a***	*10*	*a*
a	*1*	***b***	*10*	*a*
a	*1*	***b***	*20*	*b*
a	*1*	***g***	*10*	*b*
b	*2*	***a***	*10*	*a*
b	*2*	***b***	*10*	*a*
b	*2*	***b***	*20*	*b*
b	*2*	***g***	*10*	*b*

$\sigma_{A=C}(r\ x\ s)$

A	B	C	D	E
a	*1*	***a***	*10*	*a*
b	*2*	***b***	*20*	*a*
b	*2*	***b***	*20*	*b*

Banking Example

- *branch (branch-name, branch-city, assets)*
- *customer (customer-name, customer-street, customer-only)*
- *account (account-number, branch-name, balance)*
- *loan (loan-number, branch-name, amount)*
- *depositor (customer-name, account-number)*
- *borrower (customer-name, loan-number)*

Example Queries

- Find all loans of over \$1200:

  $\sigma_{amount\ >\ 1200}\ (loan)$

  – Find the loan number for each loan of an amount greater than \$1200

  $\Pi_{loan\text{-}number}\ (\sigma_{amount\ >\ 1200}\ (loan))$

- Find the names of all customers who have a loan, an account, or both, from the bank

  $\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$

  – Find the names of all customers who have a loan and an account at bank.

  $\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$

Example Queries (Cont.)

- Find the names of all customers who have a loan at the Perryridge branch.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“}Perryridge\text{”}}$$
$$(s_{\ borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower\ x\ loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name\ =\ \text{“}Perryridge\text{”}}$$
$$(\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(\text{borrower x loan}))) \ -$$
$$\Pi_{customer\text{-}name}(\text{depositor})$$

Example Queries (Cont.)

- Find the names of all customers who have a loan at the Perryridge branch.
  - Query 1

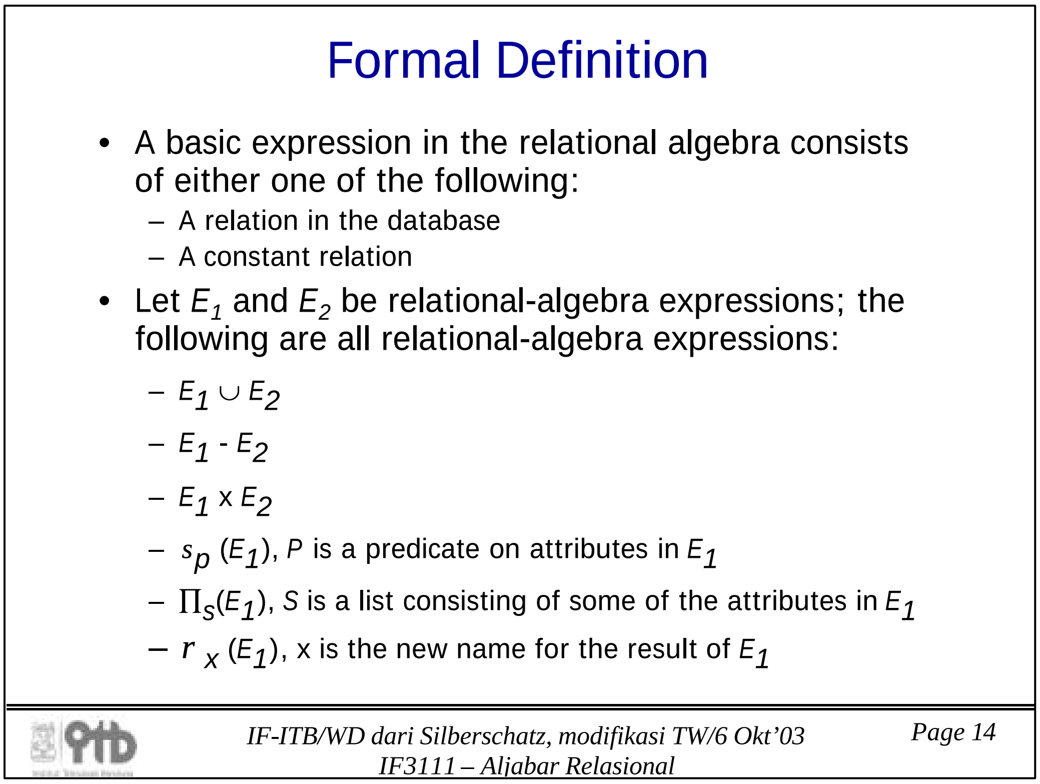
    $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}($
    $\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$

  - Query 2

    $\Pi_{customer\text{-}name}(\sigma_{loan.loan\text{-}number\ =\ borrower.loan\text{-}number}($
    $(\sigma_{branch\text{-}name\ =\ \text{“Perryridge”}}(loan)) \times borrower))$

- Find the largest account balance
  - Rename *account* relation as *d*

  $\Pi_{balance}(account)$ - $\Pi_{account.balance}$
  $(\sigma_{account.balance\ <\ d.balance}\ (account \times r_{d}\ (account)))$



Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - s_p (E_1), *P* is a predicate on attributes in E_1
  - $\Pi_S(E_1)$, *S* is a list consisting of some of the attributes in E_1
  - r_x (E_1), x is the new name for the result of E_1

Additional Operations

Some additional operations are defined that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Division
- Assignment

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

r

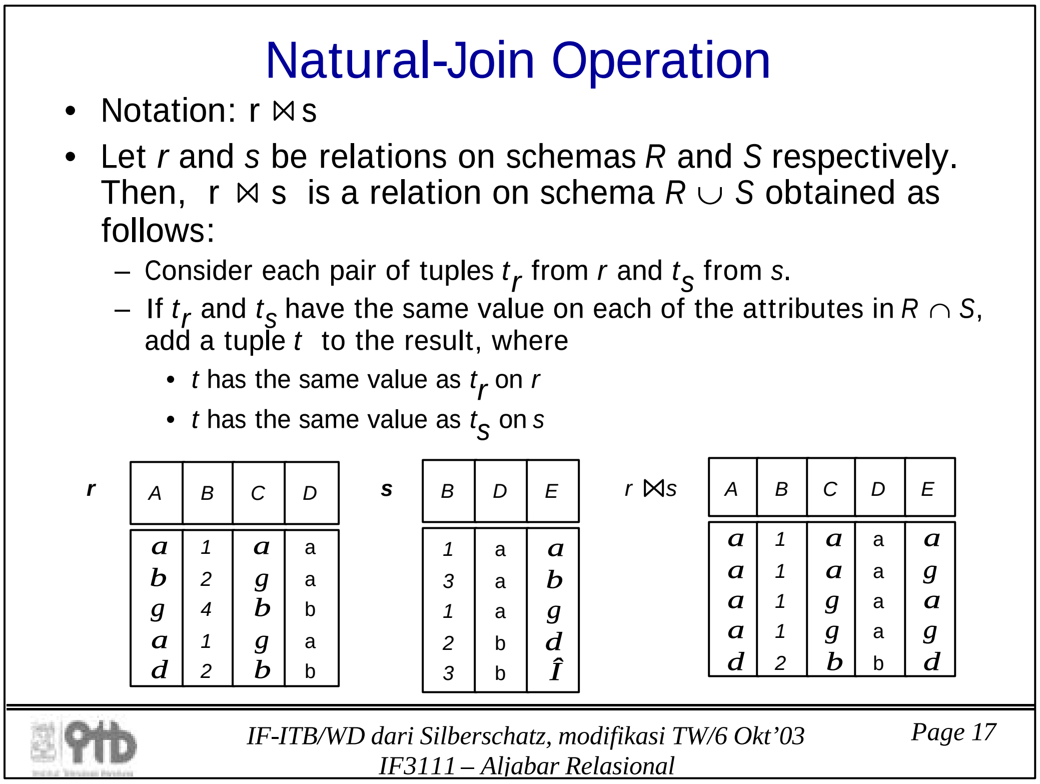
A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

r ∩ s

A	B
α	2



Natural-Join Operation

- Notation: $r \bowtie s$
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*

r

A	B	C	D
a	1	a	a
b	2	g	a
g	4	b	b
a	1	g	a
d	2	b	b

s

B	D	E
1	a	a
3	a	b
1	a	g
2	b	d
3	b	Î

$r \bowtie s$

A	B	C	D	E
a	1	a	a	a
a	1	a	a	g
a	1	g	a	a
a	1	g	a	g
d	2	b	b	d

IF-ITB/WD dari Silberschatz, modifikasi TW/6 Okt'03
IF3111 – Aljabar Relasional

Page 17

Example:

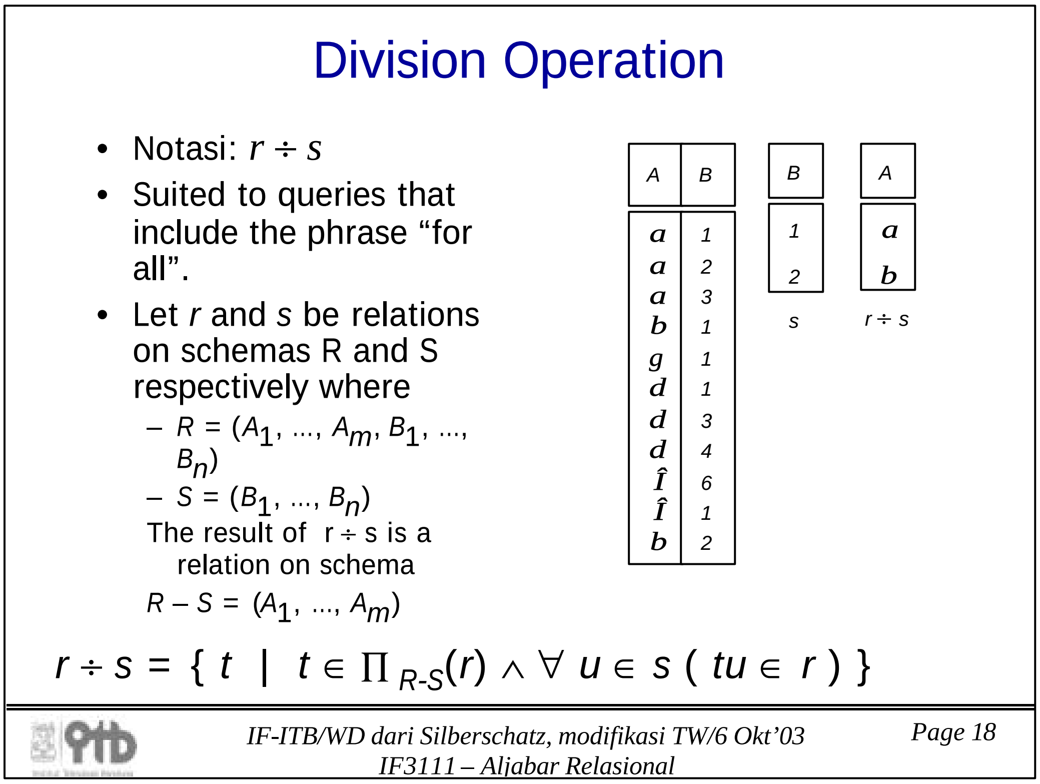
$R = (A, B, C, D)$

$S = (E, B, D)$

Result schema = (A, B, C, D, E)

$r \bowtie s$ is defined as:

$$\Pi_{r.A,\, r.B,\, r.C,\, r.D,\, s.E}\,(\sigma_{r.B = s.B \wedge r.D = s.D}\,(r \times s))$$



Division Operation

- Notasi: $r \div s$
- Suited to queries that include the phrase “for all”.
- Let *r* and *s* be relations on schemas R and S respectively where
  - $R = (A_1, ..., A_m, B_1, ..., B_n)$
  - $S = (B_1, ..., B_n)$

  The result of $r \div s$ is a relation on schema

  $R - S = (A_1, ..., A_m)$

A	B
a	1
a	2
a	3
b	1
g	1
d	1
d	3
d	4
Î	6
Î	1
b	2

B
1
2

s

A
a
b

$r \div s$

$$r \div s = \{ t \mid t \in \Pi_{R-S}(r) \wedge \forall u \in s \, (tu \in r) \}$$

Property

- Let $q - r \div s$
- Then q is the largest relation satisfying $q \times s \subseteq r$

Definition in terms of the basic algebra operation
Let *r(R)* and *s(S)* be relations, and let $S \subseteq R$

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see why

- $\Pi_{R-S,S}(r)$ simply reorders attributes of r
- $\Pi_{R-S}(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ gives those tuples t in $\Pi_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
$$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.
  - May use variable in subsequent expressions.

Example Queries

- Find all customers who have an account from at least the “Downtown” and the Uptown” branches.

Query 1

$$\Pi_{CN}(\sigma_{BN=\text{“Downtown”}}(depositor \bowtie account)) \cap$$

$$\Pi_{CN}(\sigma_{BN=\text{“Uptown”}}(depositor \bowtie account))$$

where *CN* denotes customer-name and *BN* denotes *branch-name* .

Query 2

$$\Pi_{customer\text{-}name,\ branch\text{-}name}(depositor \bowtie account)$$

$$\div\ r_{temp(branch\text{-}name)}\ (\{(\text{“Downtown”}),\ (\text{“Uptown”})\})$$

Extended Relational-Algebra-Operations

- Generalized Projection
- Outer Join
- Aggregate Functions

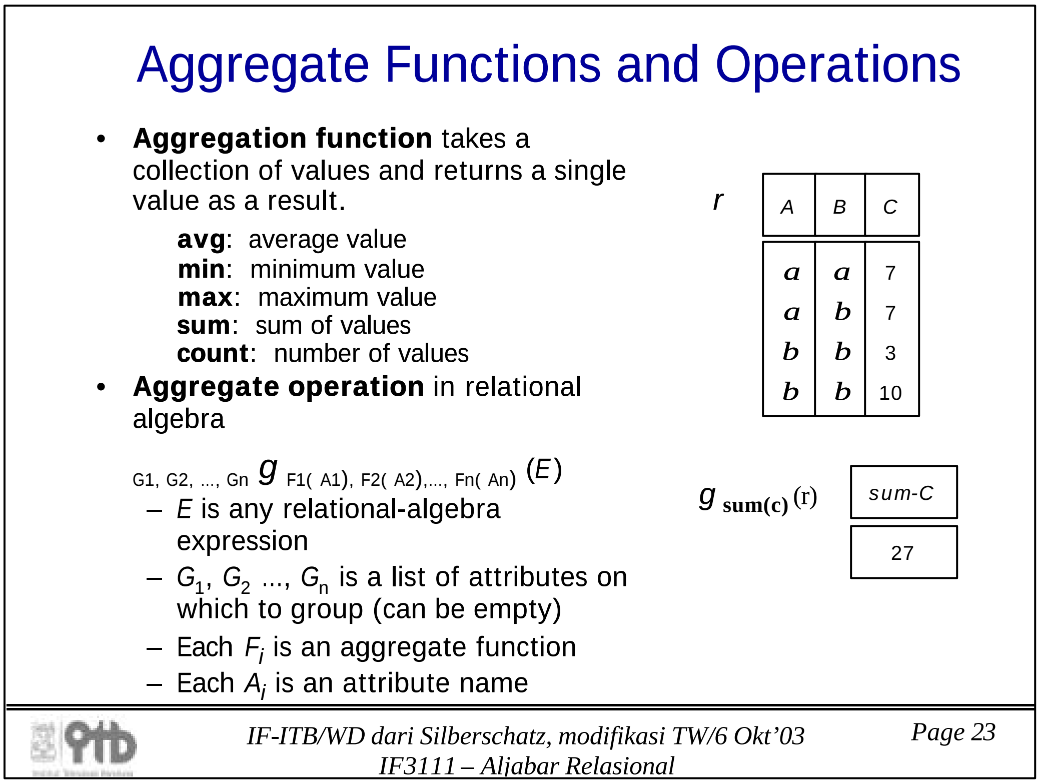
Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F1,\ F2,\ \ldots,\ Fn}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, ..., F_n are are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *credit-info(customer-name, limit, credit-balance),* find how much more each person can spend:

  $\Pi_{\textit{customer-name, limit – credit-balance}}\ (\textit{credit-info})$



Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

  $_{G1,\ G2,\ \ldots,\ Gn}\ g\ _{F1(\ A1),\ F2(\ A2),\ldots,\ Fn(\ An)}\ (E)$

  – E is any relational-algebra expression
  – G_1, G_2 ..., G_n is a list of attributes on which to group (can be empty)
  – Each F_i is an aggregate function
  – Each A_i is an attribute name

r

A	B	C
a	*a*	7
a	*b*	7
b	*b*	3
b	*b*	10

g $_{\mathbf{sum(c)}}$ (r)

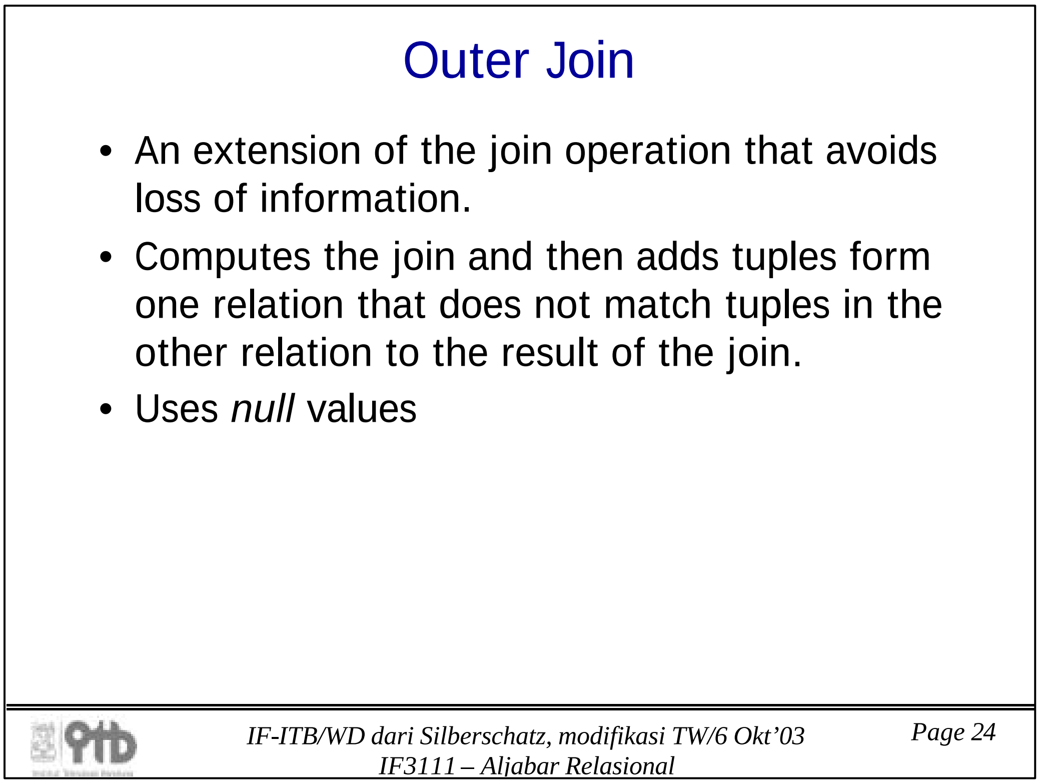
sum-C
27

IF-ITB/WD dari Silberschatz, modifikasi TW/6 Okt'03
IF3111 – Aljabar Relasional
Page 23

Result of aggregation does not have a name

Can use rename operation to give it a name

branch-name g ***sum****(balance)* **as** *sum-balance* (*account*)



Outer Join

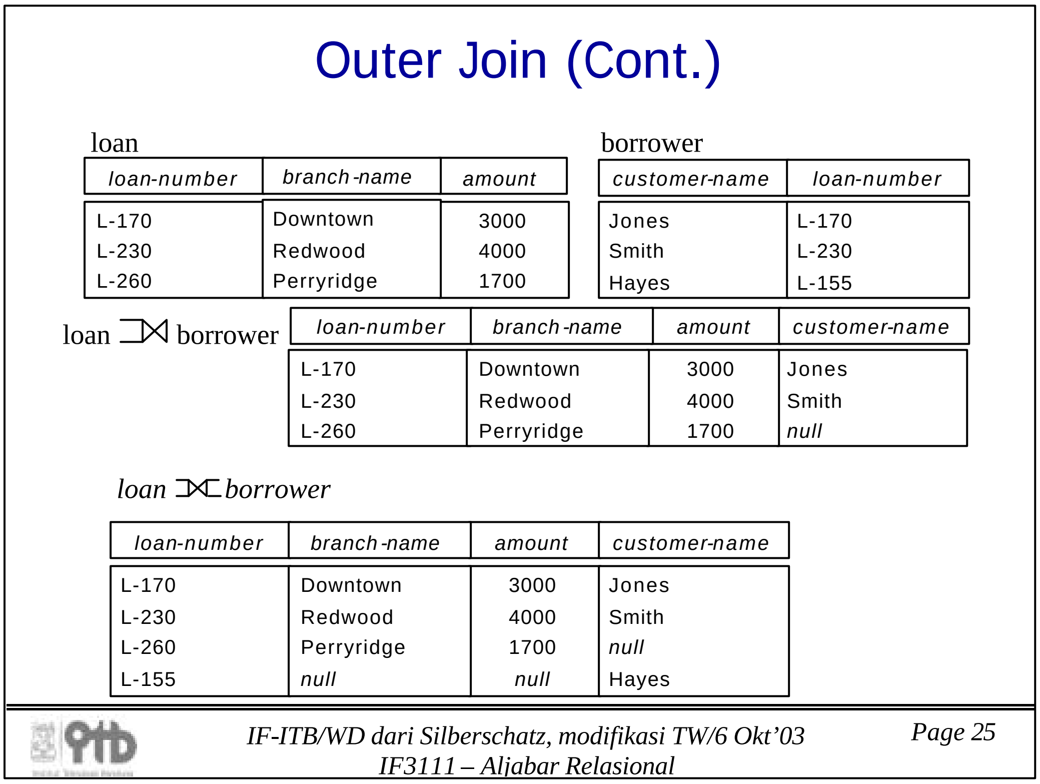
- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values

Uses *null* values:

•*null* signifies that the value is unknown or does not exist

•All comparisons involving *null* are (roughly speaking) **false** by definition.

Will study precise meaning of comparisons with nulls later



Outer Join (Cont.)

loan

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

loan ⟕ borrower

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

loan ⟗ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

IF-ITB/WD dari Silberschatz, modifikasi TW/6 Okt'03
IF3111 – Aljabar Relasional

Page 25

Contoh untuk left outer join dan full outer join

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- Comparisons with null values return the special truth value *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown,*
    (false **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*:* (**not** *unknown)* = *unknown*
  - In SQL "*P* **is unknown"** evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

  where r is a relation and E is a relational algebra query.

Delete all accounts at branches located in Needham.

$r_1 \leftarrow \sigma_{branch\text{-}city = \text{“Needham”}} (account \bowtie branch)$

$r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance} (r_1)$

$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number} (r_2 \bowtie depositor)$

$account \leftarrow account - r_2$

$depositor \leftarrow depositor - r_3$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

  $$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\text{-}name = \text{“Perryridge”}}(borrower \bowtie loan))$$

$$account \leftarrow account \cup \Pi_{branch\text{-}name,\ account\text{-}number, 200}(r_1)$$

$$depositor \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number}(r_1)$$

Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \Pi_{F1,\ F2,\ \ldots,\ Fl,}\ (r)$$

- Each F_i is either
  - the *i*th attribute of *r*, if the *i*th attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute

Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.06}(\sigma_{BAL\ >\ 10000}(account)) \cup \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(\sigma_{BAL\ £\ 10000}(account))$$

Views

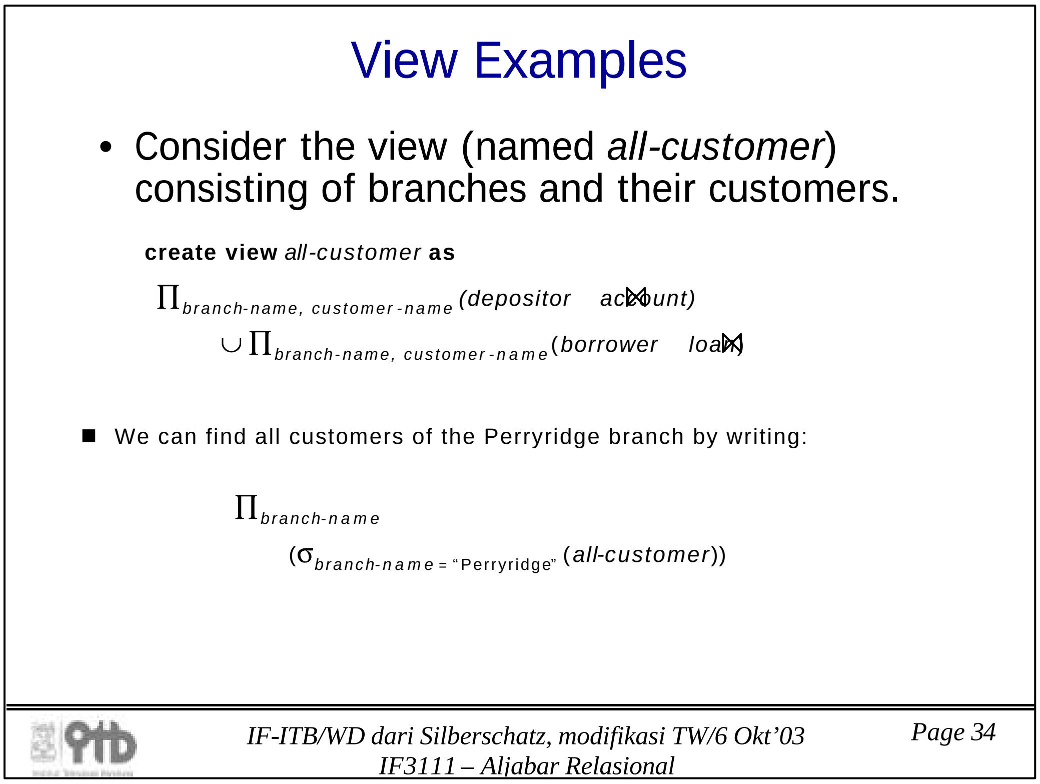
- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by
  $\Pi_{\textit{customer-name, loan-number}}(\textit{borrower} \bowtie \textit{loan})$
- Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**.

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** <query expression>

  where <query expression> is any legal relational algebra query expression. The view name is represented by *v.*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.



View Examples

- Consider the view (named *all-customer*) consisting of branches and their customers.

create view *all-customer* **as**

$$\Pi_{branch\text{-}name,\ customer\text{-}name}(depositor \bowtie account)$$
$$\cup\ \Pi_{branch\text{-}name,\ customer\text{-}name}(borrower \bowtie loan)$$

- We can find all customers of the Perryridge branch by writing:

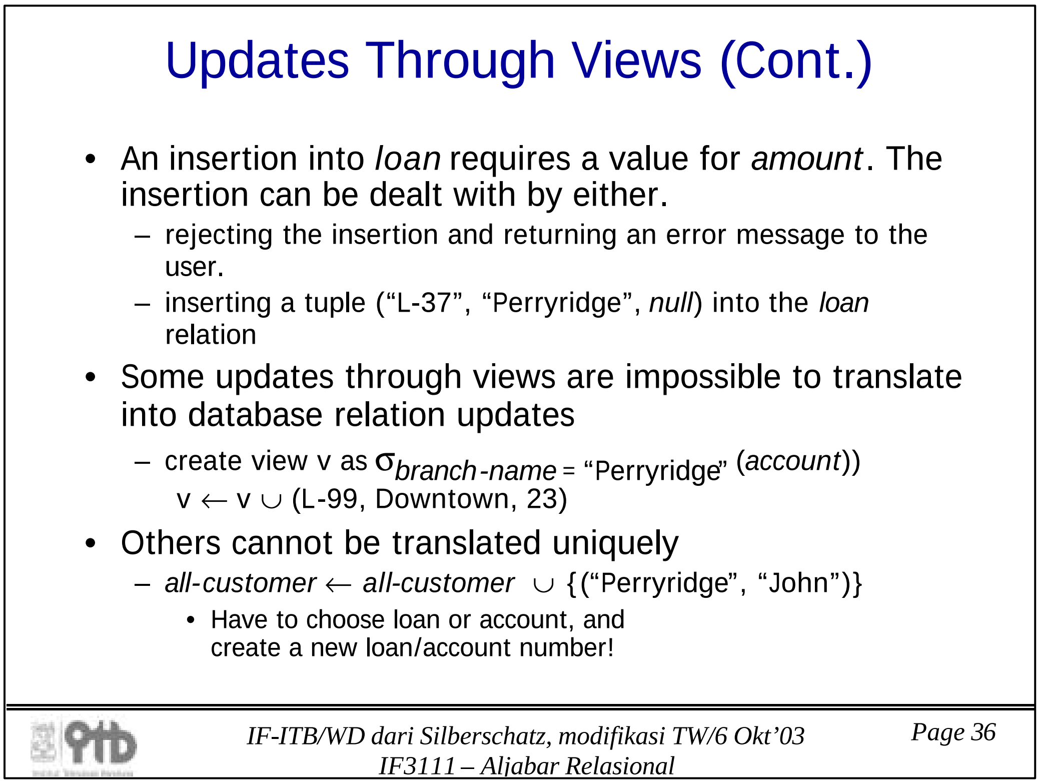
$$\Pi_{branch\text{-}name}$$
$$(\sigma_{branch\text{-}name = \text{"Perryridge"}}(all\text{-}customer))$$

Updates Through View

- Database modifications expressed as views must be translated to modifications of the actual relations in the database.
- Consider the person who needs to see all loan data in the *loan* relation except *amount.* The view given to the person, *branch-loan,* is defined as:

  create view *branch-loan* **as** $\Pi_{branch\text{-}name,\ loan\text{-}number}$ *(loan)*
- Since we allow a view name to appear wherever a relation name is allowed, the person may write:

  branch-loan $\leftarrow$ *branch-loan* $\cup$ {("Perryridge", L-37)}
- The previous insertion must be represented by an insertion into the actual relation *loan* from which the view *branch-loan* is constructed.



Updates Through Views (Cont.)

- An insertion into *loan* requires a value for *amount*. The insertion can be dealt with by either.
  - rejecting the insertion and returning an error message to the user.
  - inserting a tuple ("L-37", "Perryridge", *null*) into the *loan* relation
- Some updates through views are impossible to translate into database relation updates
  - create view v as $\sigma_{\textit{branch-name} = \text{"Perryridge"}}$ (*account*))
    v ← v ∪ (L-99, Downtown, 23)
- Others cannot be translated uniquely
  - *all-customer* ← *all-customer* ∪ {("Perryridge", "John")}
    - Have to choose loan or account, and create a new loan/account number!

One view may be used in the expression defining another view