Fundamentals of Database Systems
COMPSCI 351

Instructor: Sebastian Link

The University of Auckland

Interaction between Query Languages

A Clever Way of Writing Difficult SQL Queries

Steps

1. Formalize your query in safe relational calculus

A Clever Way of Writing Difficult SQL Queries

Steps

1. Formalize your query in safe relational calculus
2. Transform your query into SRNF

A Clever Way of Writing Difficult SQL Queries

Steps

1. Formalize your query in safe relational calculus
2. Transform your query into SRNF
3. Transform SRNF into SQL

A Clever Way of Writing Difficult SQL Queries

Steps

1. Formalize your query in safe relational calculus
2. Transform your query into SRNF
3. Transform SRNF into SQL

Example

- SRNF of the Hitchcock-query is simple to express in SQL

A Clever Way of Writing Difficult SQL Queries

Steps

1. Formalize your query in safe relational calculus
2. Transform your query into SRNF
3. Transform SRNF into SQL

Example

- SRNF of the Hitchcock-query is simple to express in SQL
- TRC queries in SRNF are even closer to SQL syntax

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathrm{n} \mid \exists d1, d1{:}\mathrm{n}(\textsc{Director}(d1) \wedge d{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists d2, d2{:}\mathrm{n}, d2{:}\mathrm{m}(\textsc{Director}(d2) \wedge d2{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists a1, a1{:}\mathrm{n}, a1{:}\mathrm{m}, a1{:}\mathrm{r}(\textsc{Actor}(a1) \wedge a1{:}\mathrm{n} = d2{:}\mathrm{n} \wedge a1{:}\mathrm{m} = d2{:}\mathrm{m} \wedge$$
$$\neg\exists a2, a2{:}\mathrm{n}, a2{:}\mathrm{m}, a2{:}\mathrm{r}(\textsc{Actor}(a2) \wedge a2{:}\mathrm{r} \neq a1{:}\mathrm{r} \wedge a2{:}\mathrm{m} = a1{:}\mathrm{m} \wedge a2{:}\mathrm{n} = a1{:}\mathrm{n}))))\}$$

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathrm{n} \mid \exists d1, d1{:}\mathrm{n}(\textsc{Director}(d1) \wedge d{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists d2, d2{:}\mathrm{n}, d2{:}\mathrm{m}(\textsc{Director}(d2) \wedge d2{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists a1, a1{:}\mathrm{n}, a1{:}\mathrm{m}, a1{:}\mathrm{r}(\textsc{Actor}(a1) \wedge a1{:}\mathrm{n} = d2{:}\mathrm{n} \wedge a1{:}\mathrm{m} = d2{:}\mathrm{m} \wedge$$
$$\neg\exists a2, a2{:}\mathrm{n}, a2{:}\mathrm{m}, a2{:}\mathrm{r}(\textsc{Actor}(a2) \wedge a2{:}\mathrm{r} \neq a1{:}\mathrm{r} \wedge a2{:}\mathrm{m} = a1{:}\mathrm{m} \wedge a2{:}\mathrm{n} = a1{:}\mathrm{n})))))\}$$

Transformed into SQL

```
SELECT d1.name FROM DIRECTOR d1
WHERE NOT EXISTS(
        SELECT d2.movie FROM DIRECTOR d2
        WHERE d2.name = d1.name AND NOT EXISTS(
                SELECT * FROM ACTOR a1
                WHERE a1.name = d2.name AND a1.movie = d2.movie
                AND NOT EXISTS(
                        SELECT * FROM ACTOR a2
                        WHERE a2.role ≠ a1.role
                          AND a2.movie = a1.movie AND a2.name= a1.name)))
```

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathsf{n} \mid \exists d1, d1{:}\mathsf{n}(\textsc{Director}(d1) \wedge d{:}\mathsf{n} = d1{:}\mathsf{n} \wedge$$
$$\neg\exists d2, d2{:}\mathsf{n}, d2{:}\mathsf{m}(\textsc{Director}(d2) \wedge d2{:}\mathsf{n} = d1{:}\mathsf{n} \wedge$$
$$\neg\exists a1, a1{:}\mathsf{n}, a1{:}\mathsf{m}, a1{:}\mathsf{r}(\textsc{Actor}(a1) \wedge a1{:}\mathsf{n} = d2{:}\mathsf{n} \wedge a1{:}\mathsf{m} = d2{:}\mathsf{m} \wedge$$
$$\neg\exists a2, a2{:}\mathsf{n}, a2{:}\mathsf{m}, a2{:}\mathsf{r}(\textsc{Actor}(a2) \wedge a2{:}\mathsf{r} \neq a1{:}\mathsf{r} \wedge a2{:}\mathsf{m} = a1{:}\mathsf{m} \wedge a2{:}\mathsf{n} = a1{:}\mathsf{n})))))\}$$

Transformed into SQL

```
SELECT d1.name FROM DIRECTOR d1
WHERE NOT EXISTS(
        SELECT d2.movie FROM DIRECTOR d2
        WHERE d2.name = d1.name AND NOT EXISTS(
                SELECT * FROM ACTOR a1
                WHERE a1.name = d2.name AND a1.movie = d2.movie
                AND NOT EXISTS(
                        SELECT * FROM ACTOR a2
                        WHERE a2.role ≠ a1.role
                          AND a2.movie = a1.movie AND a2.name= a1.name)))
```

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathrm{n} \mid \exists d1, d1{:}\mathrm{n}(\textsc{Director}(d1) \wedge d{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists d2, d2{:}\mathrm{n}, d2{:}\mathrm{m}(\textsc{Director}(d2) \wedge d2{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists a1, a1{:}\mathrm{n}, a1{:}\mathrm{m}, a1{:}\mathrm{r}(\textsc{Actor}(a1) \wedge a1{:}\mathrm{n} = d2{:}\mathrm{n} \wedge a1{:}\mathrm{m} = d2{:}\mathrm{m} \wedge$$
$$\neg\exists a2, a2{:}\mathrm{n}, a2{:}\mathrm{m}, a2{:}\mathrm{r}(\textsc{Actor}(a2) \wedge a2{:}\mathrm{r} \neq a1{:}\mathrm{r} \wedge a2{:}\mathrm{m} = a1{:}\mathrm{m} \wedge a2{:}\mathrm{n} = a1{:}\mathrm{n})))))\}$$

Transformed into SQL

```
SELECT d1.name FROM DIRECTOR d1
WHERE NOT EXISTS(
        SELECT d2.movie FROM DIRECTOR d2
        WHERE d2.name = d1.name AND NOT EXISTS(
                SELECT * FROM ACTOR a1
                WHERE a1.name = d2.name AND a1.movie = d2.movie
                AND NOT EXISTS(
                        SELECT * FROM ACTOR a2
                        WHERE a2.role ≠ a1.role
                          AND a2.movie = a1.movie AND a2.name= a1.name)))
```

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathrm{n} \mid \exists d1, d1{:}\mathrm{n}(\textsc{Director}(d1) \wedge d{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists d2, d2{:}\mathrm{n}, d2{:}\mathrm{m}(\textsc{Director}(d2) \wedge d2{:}\mathrm{n} = d1{:}\mathrm{n} \wedge$$
$$\neg\exists a1, a1{:}\mathrm{n}, a1{:}\mathrm{m}, a1{:}\mathrm{r}(\textsc{Actor}(a1) \wedge a1{:}\mathrm{n} = d2{:}\mathrm{n} \wedge a1{:}\mathrm{m} = d2{:}\mathrm{m} \wedge$$
$$\neg\exists a2, a2{:}\mathrm{n}, a2{:}\mathrm{m}, a2{:}\mathrm{r}(\textsc{Actor}(a2) \wedge a2{:}\mathrm{r} \neq a1{:}\mathrm{r} \wedge a2{:}\mathrm{m} = a1{:}\mathrm{m} \wedge a2{:}\mathrm{n} = a1{:}\mathrm{n}))))\}$$

Transformed into SQL

```
SELECT d1.name FROM DIRECTOR d1
WHERE NOT EXISTS(
         SELECT d2.movie FROM DIRECTOR d2
         WHERE d2.name = d1.name AND NOT EXISTS(
                  SELECT * FROM ACTOR a1
                  WHERE a1.name = d2.name AND a1.movie = d2.movie
                  AND NOT EXISTS(
                           SELECT * FROM ACTOR a2
                           WHERE a2.role ≠ a1.role
                              AND a2.movie = a1.movie AND a2.name= a1.name)))
```

Example: from TRC to SQL

Hitchcock query in TRC in SRNF

$$\{d{:}\mathrm{n} \mid \exists d1, d1{:}\mathrm{n}(\textsc{Director}(d1) \wedge d{:}\mathrm{n} = d1{:}\mathrm{n} \wedge \\ \neg\exists d2, d2{:}\mathrm{n}, d2{:}\mathrm{m}(\textsc{Director}(d2) \wedge d2{:}\mathrm{n} = d1{:}\mathrm{n} \wedge \\ \neg\exists a1, a1{:}\mathrm{n}, a1{:}\mathrm{m}, a1{:}\mathrm{r}(\textsc{Actor}(a1) \wedge a1{:}\mathrm{n} = d2{:}\mathrm{n} \wedge a1{:}\mathrm{m} = d2{:}\mathrm{m} \wedge \\ \neg\exists a2, a2{:}\mathrm{n}, a2{:}\mathrm{m}, a2{:}\mathrm{r}(\textsc{Actor}(a2) \wedge a2{:}\mathrm{r} \neq a1{:}\mathrm{r} \wedge a2{:}\mathrm{m} = a1{:}\mathrm{m} \wedge a2{:}\mathrm{n} = a1{:}\mathrm{n}))))\}$$

Transformed into SQL

```
SELECT d1.name FROM DIRECTOR d1
WHERE NOT EXISTS(
        SELECT d2.movie FROM DIRECTOR d2
        WHERE d2.name = d1.name AND NOT EXISTS(
                SELECT * FROM ACTOR a1
                WHERE a1.name = d2.name AND a1.movie = d2.movie
                AND NOT EXISTS(
                        SELECT * FROM ACTOR a2
                        WHERE a2.role ≠ a1.role
                          AND a2.movie = a1.movie AND a2.name= a1.name)))
```

Each relational algebra query (except union) can be easily rewritten in SQL

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

SELECT $*$ FROM R WHERE $A = B$;

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

SELECT $*$ FROM R WHERE $A = B$;

constant selection $\sigma_{A=c}(R)$

SELECT $*$ FROM R WHERE $A = c$;

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

SELECT $*$ FROM R WHERE $A = B$;

constant selection $\sigma_{A=c}(R)$

SELECT $*$ FROM R WHERE $A = c$;

projection $\pi_{A_1,\ldots,A_k}(R)$

SELECT DISTINCT $A_1, \ldots, A_k$ FROM R ;

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

$\texttt{SELECT} * \texttt{FROM}\ R\ \texttt{WHERE}\ A = B\ ;$

constant selection $\sigma_{A=c}(R)$

$\texttt{SELECT} * \texttt{FROM}\ R\ \texttt{WHERE}\ A = c\ ;$

projection $\pi_{A_1,\ldots,A_k}(R)$

$\texttt{SELECT DISTINCT}\ A_1, \ldots, A_k\ \texttt{FROM}\ R\ ;$

renaming $\delta_{A_1 \mapsto B_1,\ldots,A_k \mapsto B_k}(R)$

$\texttt{SELECT}\ A_1\ \texttt{AS}\ B_1\ , \ldots, \ A_k\ \texttt{AS}\ B_k\ \texttt{FROM}\ R\ ;$

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

`SELECT` $*$ `FROM` R `WHERE` $A = B$;

constant selection $\sigma_{A=c}(R)$

`SELECT` $*$ `FROM` R `WHERE` $A = c$;

projection $\pi_{A_1,\ldots,A_k}(R)$

`SELECT DISTINCT` $A_1, \ldots, A_k$ `FROM` R ;

renaming $\delta_{A_1 \mapsto B_1,\ldots,A_k \mapsto B_k}(R)$

`SELECT` A_1 `AS` B_1 ,..., A_k `AS` B_k `FROM` R ;

join $R_1 \bowtie R_2$ (with common attributes $A_1, \ldots, A_k$)

`SELECT` $*$ `FROM` R_1, R_2 `WHERE` $R_1.A_1 = R_2.A_1$ `AND` ... `AND` $R_1.A_k = R_2.A_k$;

SQL & Relational Algebra

Each relational algebra query (except union) can be easily rewritten in SQL

attribute selection $\sigma_{A=B}(R)$

$\texttt{SELECT} * \texttt{FROM}\ R\ \texttt{WHERE}\ A = B\ ;$

constant selection $\sigma_{A=c}(R)$

$\texttt{SELECT} * \texttt{FROM}\ R\ \texttt{WHERE}\ A = c\ ;$

projection $\pi_{A_1,\ldots,A_k}(R)$

$\texttt{SELECT DISTINCT}\ A_1, \ldots, A_k\ \texttt{FROM}\ R\ ;$

renaming $\delta_{A_1 \mapsto B_1,\ldots,A_k \mapsto B_k}(R)$

$\texttt{SELECT}\ A_1\ \texttt{AS}\ B_1\ , \ldots ,\ A_k\ \texttt{AS}\ B_k\ \texttt{FROM}\ R\ ;$

join $R_1 \bowtie R_2$ (with common attributes $A_1, \ldots, A_k$)

$\texttt{SELECT} * \texttt{FROM}\ R_1, R_2\ \texttt{WHERE}\ R_1.A_1 = R_2.A_1\ \texttt{AND} \ldots \texttt{AND}\ R_1.A_k = R_2.A_k;$

difference $R_1 - R_2$ (with attributes $A_1, \ldots, A_k$)

$\texttt{SELECT} * \texttt{FROM}\ R_1\ \texttt{WHERE}\ (A_1, \ldots, A_k)\ \texttt{NOT IN}\ R_2;$

Relational Expressions in SQL

Extend SQL by relational expressions

Relational Expressions in SQL

Extend SQL by relational expressions

Union

⟨query⟩ `UNION` ⟨query⟩

Relational Expressions in SQL

Extend SQL by relational expressions

Union

⟨query⟩ `UNION` ⟨query⟩

Intersection

⟨query⟩ `INTERSECT` ⟨query⟩

Relational Expressions in SQL

Extend SQL by relational expressions

Union	Intersection	Difference
$\langle$query$\rangle$ `UNION` $\langle$query$\rangle$	$\langle$query$\rangle$ `INTERSECT` $\langle$query$\rangle$	$\langle$query$\rangle$ `DIFFERENCE` $\langle$query$\rangle$

Relational Expressions in SQL

Extend SQL by relational expressions

Union

$\langle$query$\rangle$ `UNION` $\langle$query$\rangle$

Intersection

$\langle$query$\rangle$ `INTERSECT` $\langle$query$\rangle$

Difference

$\langle$query$\rangle$ `DIFFERENCE` $\langle$query$\rangle$

Join expressions in the `FROM`-clause

- (natural) join: R_1 `NATURAL JOIN` R_2
- equijoin: R_1 `JOIN` R_2 `ON` $R_1.A_1 = R_2.B_1$ `AND` ... `AND` $R_1.A_k = R_2.B_k$
- Θ-joins generalise equijoins by allowing inequations in the `ON`-clause

Relational Expressions in SQL

Extend SQL by relational expressions

Union

⟨query⟩ `UNION` ⟨query⟩

Intersection

⟨query⟩ `INTERSECT` ⟨query⟩

Difference

⟨query⟩ `DIFFERENCE` ⟨query⟩

Join expressions in the `FROM`-clause

- (natural) join: R_1 `NATURAL JOIN` R_2
- equijoin: R_1 `JOIN` R_2 `ON` $R_1.A_1 = R_2.B_1$ `AND` ... `AND` $R_1.A_k = R_2.B_k$
- Θ-joins generalise equijoins by allowing inequations in the `ON`-clause

All these joins in SQL except the natural join do not eliminate duplicate columns

Example for Join Expressions

List the first and last name of all movie directors of non-US movies together with the titles and production years of these movies

Example for Join Expressions

List the first and last name of all movie directors of non-US movies together with the titles and production years of these movies

```
SELECT   first_name, last_name, title, production_year
FROM     MOVIE NATURAL JOIN DIRECTOR
               NATURAL JOIN PERSON
WHERE    country <> 'USA';
```

Example for Join Expressions

List the first and last name of all movie directors of non-US movies together with the titles and production years of these movies

```
SELECT  first_name, last_name, title, production_year
FROM    MOVIE NATURAL JOIN DIRECTOR
              NATURAL JOIN PERSON
WHERE   country <> 'USA';
```

```
SELECT  p.first_name, p.last_name, m.title, m.production_year
FROM    MOVIE m JOIN DIRECTOR d ON m.title = d.title AND m.production_year = d.production_year
                JOIN PERSON p ON d.id = p.id
WHERE   m.country <> 'USA';
```

Example for Join Expressions

List the first and last name of all movie directors of non-US movies together with the titles and production years of these movies

```
SELECT  first_name, last_name, title, production_year
FROM    MOVIE NATURAL JOIN DIRECTOR
              NATURAL JOIN PERSON
WHERE   country <> 'USA';
```

```
SELECT  p.first_name, p.last_name, m.title, m.production_year
FROM    MOVIE m JOIN DIRECTOR d ON m.title = d.title AND m.production_year = d.production_year
                JOIN PERSON p ON d.id = p.id
WHERE   m.country <> 'USA';
```

Example for Join Expressions

List the first and last name of all movie directors of non-US movies together with the titles and production years of these movies

```
SELECT   first_name, last_name, title, production_year
FROM     MOVIE NATURAL JOIN DIRECTOR
               NATURAL JOIN PERSON
WHERE    country <> 'USA';
```

```
SELECT   p.first_name, p.last_name, m.title, m.production_year
FROM     MOVIE m JOIN DIRECTOR d ON m.title = d.title AND m.production_year = d.production_year
                 JOIN PERSON p ON d.id = p.id
WHERE    m.country <> 'USA';
```

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

The division operator in Relational Calculus

$\{(x_{A_1}, \ldots, x_{A_k}) \mid$

$\}$

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

The division operator in Relational Calculus

$$\{(x_{A_1}, \ldots, x_{A_k}) \mid \exists x_{B_1}, \ldots, x_{B_l}(R(x_{A_1}, \ldots, x_{A_k}, x_{B_1}, \ldots, x_{B_l}) \wedge \quad \}$$

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

The division operator in Relational Calculus

$$\{(x_{A_1}, \ldots, x_{A_k}) \mid \quad \exists x_{B_1}, \ldots, x_{B_l}(R(x_{A_1}, \ldots, x_{A_k}, x_{B_1}, \ldots, x_{B_l}) \wedge$$
$$\forall y_{B_1}, \ldots, y_{B_l}(S(y_{B_1}, \ldots, y_{B_l}) \Rightarrow$$
$$\}$$

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

The division operator in Relational Calculus

$$\begin{aligned}\{(x_{A_1}, \ldots, x_{A_k}) \mid\ & \exists x_{B_1}, \ldots, x_{B_l}(R(x_{A_1}, \ldots, x_{A_k}, x_{B_1}, \ldots, x_{B_l}) \wedge \\ & \forall y_{B_1}, \ldots, y_{B_l}(S(y_{B_1}, \ldots, y_{B_l}) \Rightarrow \\ & \exists x'_{A_1}, \ldots, x'_{A_k}, y'_{B_1}, \ldots, y'_{B_l}(R(x'_{A_1}, \ldots, x'_{A_k}, y'_{B_1}, \ldots, y'_{B_l}) \wedge \\ & \quad \}\end{aligned}$$

Division Operator: From Algebra, to Calculus, to SQL

Recall the division operator $r \div s$ where

- $\sharp r = R = \{A_1, \ldots, A_k, B_1, \ldots, B_l\}$, and
- $\sharp s = S = \{B_1, \ldots, B_l\}$

The division operator in Relational Calculus

$$
\begin{aligned}
\{(x_{A_1}, \ldots, x_{A_k}) \mid \quad & \exists x_{B_1}, \ldots, x_{B_l}(R(x_{A_1}, \ldots, x_{A_k}, x_{B_1}, \ldots, x_{B_l}) \wedge \\
& \forall y_{B_1}, \ldots, y_{B_l}(S(y_{B_1}, \ldots, y_{B_l}) \Rightarrow \\
& \exists x'_{A_1}, \ldots, x'_{A_k}, y'_{B_1}, \ldots, y'_{B_l}(R(x'_{A_1}, \ldots, x'_{A_k}, y'_{B_1}, \ldots, y'_{B_l}) \wedge \\
& (x_{A_1} = x'_{A_1}) \wedge \cdots \wedge (x_{A_k} = x'_{A_k}) \wedge (y_{B_1} = y'_{B_1}) \wedge \cdots \wedge (y_{B_l} = y'_{B_l}))))\}
\end{aligned}
$$

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid\ & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \forall y_{B_1},\ldots,y_{B_l}(S(y_{B_1},\ldots,y_{B_l}) \Rightarrow \\
& \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid \quad & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \forall y_{B_1},\ldots,y_{B_l}(S(y_{B_1},\ldots,y_{B_l}) \Rightarrow \\
& \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid\ & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \forall y_{B_1},\ldots,y_{B_l}(\neg S(y_{B_1},\ldots,y_{B_l})\vee \\
& \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$\begin{aligned}\{(x_{A_1},\ldots,x_{A_k}) \mid\ & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge\\ & \forall y_{B_1},\ldots,y_{B_l}(\neg S(y_{B_1},\ldots,y_{B_l})\vee\\ & \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge\\ & (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}\end{aligned}$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid \quad & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \neg\exists y_{B_1},\ldots,y_{B_l}\neg(\neg S(y_{B_1},\ldots,y_{B_l})\vee \\
& \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid\ & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \neg\exists y_{B_1},\ldots,y_{B_l}\neg(\neg S(y_{B_1},\ldots,y_{B_l})\vee \\
& \exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l})))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid \quad & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \neg\exists y_{B_1},\ldots,y_{B_l}(\neg\neg S(y_{B_1},\ldots,y_{B_l})\wedge \\
& \neg\exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid \quad & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \neg\exists y_{B_1},\ldots,y_{B_l}(\neg\neg S(y_{B_1},\ldots,y_{B_l})\wedge \\
& \neg\exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$\begin{array}{ll} \{(x_{A_1},\ldots,x_{A_k}) \mid & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\ & \neg\exists y_{B_1},\ldots,y_{B_l}(S(y_{B_1},\ldots,y_{B_l})\wedge \\ & \neg\exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\ & (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\} \end{array}$$

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$\{(x_{A_1},\ldots,x_{A_k}) \mid \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge$$
$$\neg\exists y_{B_1},\ldots,y_{B_l}(S(y_{B_1},\ldots,y_{B_l})\wedge$$
$$\neg\exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge$$
$$(x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}$$

This gives us an SQL implementation of the division operator

```
SELECT R1.A1, ..., R1.Ak
FROM R AS R1
WHERE NOT EXISTS (SELECT *
                  FROM S
                  WHERE NOT EXISTS (
                              SELECT *
                              FROM R AS R2
                              WHERE R2.A1 = R1.A1 AND ... AND R2.Ak = R1.Ak AND
                                    R2.B1 = S.B1 AND ... AND R2.Bl = S.Bl))
```

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$\{(x_{A_1},\dots,x_{A_k}) \mid \exists x_{B_1},\dots,x_{B_l}(R(x_{A_1},\dots,x_{A_k},x_{B_1},\dots,x_{B_l})\wedge$$
$$\neg\exists y_{B_1},\dots,y_{B_l}(S(y_{B_1},\dots,y_{B_l})\wedge$$
$$\neg\exists x'_{A_1},\dots,x'_{A_k},y'_{B_1},\dots,y'_{B_l}(R(x'_{A_1},\dots,x'_{A_k},y'_{B_1},\dots,y'_{B_l})\wedge$$
$$(x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}$$

This gives us an SQL implementation of the division operator

```
SELECT R1.A1, ..., R1.Ak
FROM R AS R1
WHERE NOT EXISTS (SELECT *
                  FROM S
                  WHERE NOT EXISTS (
                                    SELECT *
                                    FROM R AS R2
                                    WHERE R2.A1 = R1.A1 AND ... AND R2.Ak = R1.Ak AND
                                          R2.B1 = S.B1 AND ... AND R2.Bl = S.Bl))
```

Division Operator: From Algebra, to Calculus, to SQL

Division operator in Relational Calculus using SRNF

$$
\begin{aligned}
\{(x_{A_1},\ldots,x_{A_k}) \mid\ & \exists x_{B_1},\ldots,x_{B_l}(R(x_{A_1},\ldots,x_{A_k},x_{B_1},\ldots,x_{B_l})\wedge \\
& \neg\exists y_{B_1},\ldots,y_{B_l}(S(y_{B_1},\ldots,y_{B_l})\wedge \\
& \neg\exists x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l}(R(x'_{A_1},\ldots,x'_{A_k},y'_{B_1},\ldots,y'_{B_l})\wedge \\
& (x_{A_1}=x'_{A_1})\wedge\cdots\wedge(x_{A_k}=x'_{A_k})\wedge(y_{B_1}=y'_{B_1})\wedge\cdots\wedge(y_{B_l}=y'_{B_l}))))\}
\end{aligned}
$$

This gives us an SQL implementation of the division operator

SELECT $R_1.A_1,\ldots,R_1.A_k$
FROM R AS R_1
WHERE NOT EXISTS (SELECT $*$
FROM S
WHERE NOT EXISTS (
SELECT $*$
FROM R AS R_2
WHERE $R_2.A_1 = R_1.A_1$ AND ... AND $R_2.A_k = R_1.A_k$ AND
$R_2.B_1 = S.B_1$ AND ... AND $R_2.B_l = S.B_l$))

Division Operator in SQL: No Nesting, but Counting

$$
\begin{array}{ll}
\texttt{SELECT} & A_1,\ldots,A_k \\
\texttt{FROM} & R \\
\texttt{WHERE} & B_1,\ldots,B_l\ \texttt{IN}\ (\texttt{SELECT}\ B_1,\ldots,B_l\ \texttt{FROM}\ S) \\
\texttt{GROUP BY} & A_1,\ldots,A_k \\
\texttt{HAVING} & \texttt{COUNT}(*){=}(\texttt{SELECT COUNT}(*)\ \texttt{FROM}\ S)
\end{array}
$$

Division Operator in SQL: No Nesting, but Counting

```
SELECT      A_1, ..., A_k
FROM        R
WHERE       B_1, ..., B_l IN (SELECT B_1, ..., B_l FROM S)
GROUP BY    A_1, ..., A_k
HAVING      COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess
Gill	Badminton

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

```
SELECT     A1, ..., Ak
FROM       R
WHERE      B1, ..., Bl IN (SELECT B1, ..., Bl FROM S)
GROUP BY   A1, ..., Ak
HAVING     COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess
Gill	Badminton

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

```
SELECT     A1, ..., Ak
FROM       R
WHERE      B1, ..., Bl IN (SELECT B1, ..., Bl FROM S)
GROUP BY   A1, ..., Ak
HAVING     COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

$$
\begin{array}{ll}
\texttt{SELECT} & A_1, \ldots, A_k \\
\texttt{FROM} & R \\
\texttt{WHERE} & B_1, \ldots, B_l \ \texttt{IN} \ (\texttt{SELECT} \ B_1, \ldots, B_l \ \texttt{FROM} \ S) \\
\color{red}{\texttt{GROUP BY}} & \color{red}{A_1, \ldots, A_k} \\
\texttt{HAVING} & \texttt{COUNT}(*)\!=\!(\texttt{SELECT COUNT}(*) \ \texttt{FROM} \ S)
\end{array}
$$

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

$$
\begin{array}{ll}
\texttt{SELECT} & A_1, \ldots, A_k \\
\texttt{FROM} & R \\
\texttt{WHERE} & B_1, \ldots, B_l \ \texttt{IN} \ (\texttt{SELECT} \ B_1, \ldots, B_l \ \texttt{FROM} \ S) \\
\color{red}{\texttt{GROUP BY}} & \color{red}{A_1, \ldots, A_k} \\
\texttt{HAVING} & \texttt{COUNT}(*){=}(\texttt{SELECT COUNT}(*) \ \texttt{FROM} \ S)
\end{array}
$$

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

```
SELECT      A1, ..., Ak
FROM        R
WHERE       B1, ..., Bl IN (SELECT B1, ..., Bl FROM S)
GROUP BY    A1, ..., Ak
HAVING      COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

```
SELECT     A1, ..., Ak
FROM       R
WHERE      B1, ..., Bl IN (SELECT B1, ..., Bl FROM S)
GROUP BY   A1, ..., Ak
HAVING     COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

```
SELECT     A1, ..., Ak
FROM       R
WHERE      B1, ..., Bl IN (SELECT B1, ..., Bl FROM S)
GROUP BY   A1, ..., Ak
HAVING     COUNT(*)=(SELECT COUNT(*) FROM S)
```

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

SELECT	$A_1, \ldots, A_k$
FROM	R
WHERE	$B_1, \ldots, B_l$ IN (SELECT $B_1, \ldots, B_l$ FROM S)
GROUP BY	$A_1, \ldots, A_k$
HAVING	COUNT(∗)=(SELECT COUNT(∗) FROM S)

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person

Division Operator in SQL: No Nesting, but Counting

$$\begin{array}{ll}
\texttt{SELECT} & A_1, \ldots, A_k \\
\texttt{FROM} & R \\
\texttt{WHERE} & B_1, \ldots, B_l \ \texttt{IN} \ (\texttt{SELECT} \ B_1, \ldots, B_l \ \texttt{FROM} \ S) \\
\texttt{GROUP BY} & A_1, \ldots, A_k \\
\texttt{HAVING} & \texttt{COUNT}(*) = (\texttt{SELECT COUNT}(*) \ \texttt{FROM} \ S)
\end{array}$$

R

Person	Hobby
Jack	Tennis
Gill	Tennis
Gill	Chess

S

Hobby
Tennis
Chess

$R \div S$

Person
Gill

Outer Joins

Dangling tuples

- Tuples $t_1 \in r_1$ not matching tuples $t_2 \in r_2$ are not in the join $r_1 \bowtie r_2$
- The same applies to the equijoin and general Θ-joins

Outer Joins

Dangling tuples

- Tuples $t_1 \in r_1$ not matching tuples $t_2 \in r_2$ are not in the join $r_1 \bowtie r_2$
- The same applies to the equijoin and general Θ-joins

Left Outer Join

For dangling tuples $t_1 \in r_1$ fill up the additional attributes from $\#r_2$ with nulls

Outer Joins

Dangling tuples

- Tuples $t_1 \in r_1$ not matching tuples $t_2 \in r_2$ are not in the join $r_1 \bowtie r_2$
- The same applies to the equijoin and general Θ-joins

Left Outer Join

For dangling tuples $t_1 \in r_1$ fill up the additional attributes from $\#r_2$ with nulls

Right Outer Join

For dangling tuples $t_2 \in r_1$ fill up the additional attributes from $\#r_1$ with nulls

Outer Joins

Dangling tuples

- Tuples $t_1 \in r_1$ not matching tuples $t_2 \in r_2$ are not in the join $r_1 \bowtie r_2$
- The same applies to the equijoin and general Θ-joins

Left Outer Join

For dangling tuples $t_1 \in r_1$ fill up the additional attributes from $\# r_2$ with nulls

Right Outer Join

For dangling tuples $t_2 \in r_1$ fill up the additional attributes from $\# r_1$ with nulls

Full Outer Join

Combine rules for left and right outer joins

Outer Joins

Dangling tuples

- Tuples $t_1 \in r_1$ not matching tuples $t_2 \in r_2$ are not in the join $r_1 \bowtie r_2$
- The same applies to the equijoin and general Θ-joins

Left Outer Join

For dangling tuples $t_1 \in r_1$ fill up the additional attributes from $\# r_2$ with nulls

Right Outer Join

For dangling tuples $t_2 \in r_1$ fill up the additional attributes from $\# r_1$ with nulls

Full Outer Join

Combine rules for left and right outer joins

Extension

Similar for equijoins and Θ-joins

Example: Full Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

Example: Full Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

The natural join $r \bowtie s$

cinema	*movie*	*date*	*time*	*name*
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Example: Full Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

The natural join $r \bowtie s$

cinema	*movie*	*date*	*time*	*name*
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

r FULL OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
Downtown	The Godfather	02.04.	8:30pm	NULL
NULL	The Departed	NULL	NULL	M. Scorsese
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Example: Full Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

The natural join $r \bowtie s$

cinema	*movie*	*date*	*time*	*name*
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

r FULL OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
Downtown	The Godfather	02.04.	8:30pm	NULL
NULL	The Departed	NULL	NULL	M. Scorsese
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Example: Full Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

The natural join $r \bowtie s$

cinema	*movie*	*date*	*time*	*name*
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

r FULL OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
Downtown	The Godfather	02.04.	8:30pm	NULL
NULL	The Departed	NULL	NULL	M. Scorsese
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Example: Left and Right Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

Example: Left and Right Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

r LEFT OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
Downtown	The Godfather	02.04.	8:30pm	NULL
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Example: Left and Right Outer Join

Consider the following two relations r and s

cinema	*movie*	*date*	*time*
Downtown	The Godfather	02.04.	8:30pm
Downtown	The Seven Samurai	02.04.	8pm

movie	*name*
The Departed	M. Scorsese
The Seven Samurai	Akira Kurosawa

r LEFT OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
Downtown	The Godfather	02.04.	8:30pm	NULL
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

r RIGHT OUTER JOIN s

cinema	*movie*	*date*	*time*	*name*
NULL	The Departed	NULL	NULL	M. Scorsese
Downtown	The Seven Samurai	02.04.	8pm	Akira Kurosawa

Outer Joins in SQL

Syntax for outer join expressions in SQL (in `FROM`-clauses)

R_1 [`FULL` | `LEFT` | `RIGHT`] `OUTER JOIN` R_2 [`ON` ⟨join-condition⟩]

Outer Joins in SQL

Syntax for outer join expressions in SQL (in `FROM`-clauses)

R_1 [`FULL` | `LEFT` | `RIGHT`] `OUTER JOIN` R_2 [`ON` ⟨join-condition⟩]

full outer join $r_1[\bowtie]r_2$ **of relations r_1 and r_2 is defined as**

$$
\begin{aligned}
&\{t \mid \exists t_1 \in r_1, t_2 \in r_2 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2] = t_2[\#r_2]\}\\
\cup&\{t \mid \exists t_1 \in r_1 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2 - \#r_1] = \mathrm{NULL} \wedge \neg\exists t_2 \in r_2 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\}\\
\cup&\{t \mid \exists t_2 \in r_2 . t[\#r_2] = t_2[\#r_2] \wedge t[\#r_1 - \#r_2] = \mathrm{NULL} \wedge \neg\exists t_1 \in r_1 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\}
\end{aligned}
$$

Outer Joins in SQL

Syntax for outer join expressions in SQL (in `FROM`-clauses)

R_1 [`FULL` | `LEFT` | `RIGHT`] `OUTER JOIN` R_2 [`ON` ⟨join-condition⟩]

full outer join $r_1[\bowtie]r_2$ **of relations r_1 and r_2 is defined as**

$$\begin{aligned}
&\{t \mid \exists t_1 \in r_1, t_2 \in r_2 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2] = t_2[\#r_2]\}\\
\cup&\{t \mid \exists t_1 \in r_1 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2 - \#r_1] = \text{NULL} \wedge \neg\exists t_2 \in r_2 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\}\\
\cup&\{t \mid \exists t_2 \in r_2 . t[\#r_2] = t_2[\#r_2] \wedge t[\#r_1 - \#r_2] = \text{NULL} \wedge \neg\exists t_1 \in r_1 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\}
\end{aligned}$$

left outer join **of relations r_1 and r_2**

omit the last of these three sets in the union

Outer Joins in SQL

Syntax for outer join expressions in SQL (in `FROM`-clauses)

R_1 [`FULL` | `LEFT` | `RIGHT`] `OUTER JOIN` R_2 [`ON` ⟨join-condition⟩]

full outer join $r_1[\bowtie]r_2$ **of relations r_1 and r_2 is defined as**

$$
\begin{aligned}
&\{t \mid \exists t_1 \in r_1, t_2 \in r_2 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2] = t_2[\#r_2]\} \\
\cup&\{t \mid \exists t_1 \in r_1 . t[\#r_1] = t_1[\#r_1] \wedge t[\#r_2 - \#r_1] = \mathrm{NULL} \wedge \neg\exists t_2 \in r_2 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\} \\
\cup&\{t \mid \exists t_2 \in r_2 . t[\#r_2] = t_2[\#r_2] \wedge t[\#r_1 - \#r_2] = \mathrm{NULL} \wedge \neg\exists t_1 \in r_1 . t_1[\#r_1 \cap \#r_2] = t_2[\#r_1 \cap \#r_2]\}
\end{aligned}
$$

left outer join **of relations r_1 and r_2**

omit the last of these three sets in the union

right outer join **of relations r_1 and r_2**

omit the second of these three sets in the union

Summary

- The core of SQL is equivalent to relational algebra and relational calculus
- Relational calculus can help in writing complex SQL queries
- Relational algebra is used to speed up query evaluation
- More sophisticated commands help with query formulation