# Lecture 6 Relational Algebra ===============================

## 1. Operators

### 1.1 Selection (= where clause in sql)

**Selection Operator**

$\sigma_{[c]}(R)$ **selects** all rows from the relation $R$ that satisfies the selection condition $c$*.

💡 Although it is called 'selection', it actually maps to 'where' clause in sql.

▼ Examples

- Find the name (rname) and area of the different restaurants in London.

$$\sigma[\text{area} = \text{'London'}](\text{restaurant})$$

```
select * from restaurant r
where r.area = 'London'
```

### 1.2 Projection (= select clause in sql)

**Projection Operator**

$\pi_{[l]}(R)$ **keeps** only the columns specified in the ordered list $l$ and in the same order*.

▼ Example

- Find the different name (cname) of customers that like at least one pizza.

$$\pi[\text{cname}](\text{likes})$$

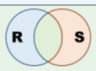
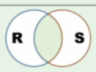
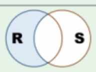
```
SELECT l.cname
FROM likes l;
```

### 1.3 Renaming

**Renaming Operator**

$\rho_{[r]}(R)$ **renames** all the attributes mentioned in $r$*.

### 1.4 Set Operations

| Operation | Visualization | SQL |
|---|---|---|
| $R \cup S$ |  | SELECT * FROM R<br>UNION<br>SELECT * FROM S |
| $R \cap S$ |  | SELECT * FROM R<br>INTERSECT<br>SELECT * FROM S |
| $R - S$ |  | SELECT * FROM R<br>EXCEPT<br>SELECT * FROM S |

The two relations must be **union-compatible** *(basically, they must have the same column types)*.

▼ Examples

Find the different pizza sold by both Bella Italia and Desert Diner.

$$Q1 := \pi[\text{pizza}](\sigma[\text{rname} = \text{'Bella Italia'}](\text{sells}))$$
$$Q2 := \pi[\text{pizza}](\sigma[\text{rname} = \text{'Desert Diner'}](\text{sells}))$$
$$Q1 \cap Q2$$

### 1.5 Cross Product (=Cartesian Products)

**Product**
Operator

FROM clause

**Cross Product** (Cartesian Product)

$R_1 \times R_2$ combine each row of $R_1$ with each row of $R_2$ and keep the $n$ columns of $R_1$ and the $m$ columns of $R_2$.

$R_1$ (2)

| a | b |
|---|---|
| 1 | 2 |
| 3 | 4 |

$R_2$ (3)

| c | d | e |
|---|---|---|
| A | B | C |
| D | E | F |
| G | H | I |

$R_1 \times R_2$ (6)

| a | b | c | d | e |
|---|---|---|---|---|
| 1 | 2 | A | B | C |
| 1 | 2 | D | E | F |
| 1 | 2 | G | H | I |
| 3 | 4 | A | B | C |
| 3 | 4 | D | E | F |
| 3 | 4 | G | H | I |

### 1.6 Join (on condition c)

**Join**

$R_1 \bowtie_{[c]} R_2$ is simply defined as $\sigma_{[c]}(R_1 \times R_2)$. In other words, we include only tuples that satisfies the condition $c$ after the cross product.

**SELECT-FROM-WHERE**

```
SELECT a1, a2, a3, ...
FROM r1, r2, r3, ...
WHERE c;
```

```
π[a1, a2, a3, ...](
  σ[c](r1 x r2 x r3 x ...)
)
```

**Two Relations ; Distinct Attributes**

```
SELECT a1, a2, a3, ...
FROM r1, r2
WHERE c;
```

```
π[a1, a2, a3, ...](
  r1 ⋈[c] r2
)
```

## 2. Examples

▼ Examples

- Find all the different pairs of customer name and restaurant name such that they are in the same area.

**Relational Algebra**

```
π[cname, rname](
  customer ⋈[area = rarea]
     ρ[rarea <- area](restaurant)
)
```

**Note**

Still need renaming.

**Result**

| cname | rname |
|---|---|
| Alice | Bella Italia |
| Alice | Big Apple Bistro |
| Alice | Down Under Delights |
| ... | |

*28 rows*

## 3. Writing Conventions

**Extension**
Written

**Extended Algebra**

For written algebra, we add the following capabilities to our relational algebra.

- **Relation Renaming** — $\rho_{[R_2]}(R_1)$ renames relation $R_1$ into $R_2$.
- **Dot Notation** — $R.attr$ refers to the attribute $attr$ of relation $R$ which may come from a renamed relation.

```
SELECT r.a1, s.a2
FROM rel1 r, rel2 s
WHERE c;
```

⟺

```
π[r.a1, s.a2](
  σ[c](
    ρ[r](rel1) x ρ[s](rel2)))
```

ρ[r](rel1)

# Lecture 8 Functional Dependencies========================

## 1. Closure Algorithm

**Algorithm #1: Attribute Closure**

**input** $S, \Sigma$
**output** $S^+$

**begin**
  $\Omega := \Sigma$; // $\Omega$ stands for "unused"
  $\Gamma := S$; // $\Gamma$ stands for "closure"
  **while** $(X \to Y \in \Omega)$ *and* $(X \subseteq \Gamma)$ **do**
    $\Omega := \Omega - \{X \to Y\}$;
    $\Gamma := \Gamma \cup Y$;
  **return** $\Gamma$

## 2. Trivialities

### 2.1 Trivial

**Definition**

An fd $\sigma : X \to Y$ is ***trivial*** if and only if $Y \subseteq X$.

Let $R = \{A, B, C\}$

- $\{A\} \to \{A\}$ is trivial
- $\{A,B\} \to \{A\}$ is trivial
- $\{A,B\} \to \{\}$ is trivial *(also denoted as $\{A,B\} \to \varnothing$)*

### 2.2 Non-Trivial

**Triviality**
Non-Trivial

**Definition**

An fd $\sigma : X \to Y$ is ***non-trivial*** if and only if $Y \not\subseteq X$.

Superset/ Disjoint

Let $R = \{A, B, C\}$

- $\{A\} \to \{B\}$ is non-trivial
- $\{A,C\} \to \{B,C\}$ is non-trivial
- $\{\} \to \{A,B\}$ is completely non-trivial *(also denoted as $\varnothing \to \{A,B\}$)*

Must be a constant

### 2.3 Completely non-trivial

**Completely**

Disjoint set

**Definition**

An fd $\sigma : X \to Y$ is ***completely non-trivial*** if and only if $Y \neq \varnothing$ and $Y \cap X = \varnothing$.

Let $R = \{A, B, C\}$

Only split the RHS
Never LHS

- $\{A\} \to \{B\}$ is completely non-trivial
- $\{A,C\} \to \{B,C\}$ is not completely non-trivial

⇒ Completely non-trivial $\{A,C\} \to \{B\}$.
⇒ Trivial $\{A,C\} \to \{C\}$

**Theorem #1**

A non-trivial *(but not completely non-trivial)* functional dependency can be split into a trivial functional dependency and a completely non-trivial functional dependency.

## 3. Keys & other definitions

### 3.1 Super Key

Using **more than enough columns** to **uniquely identify tuples** in a table (CPT103)

**Definition**

Let $R$ be a relation. Let $S \subseteq R$ be a set of attributes of $R$. $S$ is a **superkey** of $R$ if and only if $S \to R$.

**In Other Words**

A **superkey** is a set of attributes of a relation whose knowledge determines the value of the entire tuple.

**Theorem #2**

A relation $R$ have *at least 1* superkey. $R \to R$

### 3.2 Candidate Key

All these possible primary keys are called candidate keys. The PK is just a CK chosen by the table designer. (CPT103)

**In Other Words**

A **candidate key** is a *minimal* superkey.
The **primary key** is the candidate key that the designed prefers.

### 3.3 Compare and contrast

**Super key:** Collection of PK (**NON**-minimal)
**Candidate key:** Collection of PK that **must be minimal** (you **can't remove any of them**)

### 3.4 Prime Attributes (Lecture 1)

**Each column composing the candidate key** is said to be a prime attribute.
E.g. For candidate keys AC, CD, CE, the prime attributes are ACDE.

## 4. Closures

**Closure**
Closure of $\Sigma$


$\{A\} \to \{B\}$
$\{C\} \to \{A\}$
$\{C\} \to \{B\}$

**Definition**

Let $\Sigma$ be a set of functional dependencies of a relation schema $R$. The ***closure*** of $\Sigma$ --denoted by $\Sigma^+$-- is the set of all functional dependencies *logically* entailed by the functional dependencies in $\Sigma$.

## 5. Armstrong Rules

**Armstrong**
Rules

Reflexivity　Augmentation　if $\{A\} \to \{B\}$

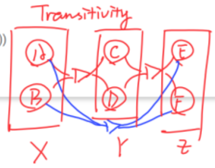

**Armstrong Axioms**

$\{A, B\}$ X　$Y \subseteq \{A\}$　then X　Y $\{B, C\}$

Let $R$ be a set of attributes. The following inference rules are the ***Armstrong Axiom***.

- **Reflexivity**
  $\forall X \subseteq R\ \forall Y \subseteq R\ ((Y \subseteq X) \Rightarrow (X \to Y))$
- **Augmentation**
  $\forall X \subseteq R\ \forall Y \subseteq R\ \forall Z \subseteq R\ ((X \to Y) \Rightarrow ((X \cup Z) \to (Y \cup Z)))$
- **Transitivity**
  $\forall X \subseteq R\ \forall Y \subseteq R\ \forall Z \subseteq R\ ((X \to Y \wedge Y \to Z) \Rightarrow (X \to Z))$

Transitivity




if $\{A,B\} \to \{C,D\}$
and $\{C,D\} \to \{E,F\}$
then $\{A,B\} \to \{E,F\}$

## 6. Minimal Cover and Canonical Cover Algorithm

**Algorithm #3: Canonical Cover**

**input** $\Sigma$
**output** $\Sigma_4$

1. Simplify *(minimize)* the right hand side of every functional dependency in $\Sigma$ produce $\Sigma_1$
2. Simplify *(minimize)* the left hand side of every functional dependency in $\Sigma_1$ produce $\Sigma_2$
3. Simplify *(minimize)* the set $\Sigma_2$ to produce $\Sigma_3$
4. Regroup all functional dependencies with the same left hand side in $\Sigma_3$ produce $\Sigma_4$
5. Return $\Sigma_4$

**Step #2: Simplifying Left Hand Side**

Let $X \to \{A\}$ be a functional dependency in $\Sigma'$. Attribute $B \in X$ can be removed from $X$ if

$(X - \{B\}) \to \{A\}$ is **logically entailed** by $\Sigma'$

Then we can replace $X \to Y$ by $(X - \{B\}) \to \{A\}$ in $\Sigma'$

**Step #3: Simplifying the Set**

Let $X \to Y$ be a functional dependency in $\Sigma'$. It can be removed from $\Sigma'$ if

$X \to Y$ is **logically entailed** by $(\Sigma' - \{X \to Y\})$

Then we can replace $\Sigma$ by $(\Sigma' - \{X \to Y\})$.

# Lecture 9 Boyce-Codd Normal Form ==========================

## 1. Definition & Checking

**Boyce-Codd Normal Form**

A relation $R$ with a set of functional dependencies $\Sigma$ is in **BCNF** if and only if for every functional dependency $X \to \{A\} \in \Sigma^+$:

- $X \to \{A\}$ is trivial, or
- $X$ is a superkey

**Note**

For relation $R$ **before decomposition**, it is sufficient only to look at $\Sigma$.

Check for violation. find $X \to \{A\}$ s.t.
① $X \to \{A\}$ is non-trivial and
② $X$ is not a super key

*Although this is a theorem, we omit the proof so we do not give it a number.

## 2. Lossless Join & Checking

**Lossless-Join Decomposition**

A binary decomposition is ***lossless-join*** if and only if the ~~full~~ outer natural join of its two **fragments** *(i.e., the two tables resulting from the decomposition)* equals the initial table. Otherwise, the decomposition is **lossy**.

Input: $R_1, R_2$
① Find $R_1 \cap R_2$, call this $R_n$
② compute $R_n^+$ w.r.t $\Sigma$
③ Either or both $R_1 \subseteq R_n^+$, $R_2 \subseteq R_n^+$
$\Rightarrow$ Binary lossless join

**Lossless-Join**
Lemma

**Lemma #1: Lossless-Join Binary Decomposition**

*A binary decomposition of $R$ into $R_1$ and $R_2$ is* **lossless-join** if $R = R_1 \cup R_2$ and $(R_1 \cap R_2) \to R_1$ or $(R_1 \cap R_2) \to R_2$

## 3. Projection of Functional Dependencies

**Projected Functional Dependencies**

Consider a relation $R$ with a set of functional dependencies $\Sigma$. A set ~~$Sigma'$~~ $\Sigma'$ of ***projected functional dependencies*** on $R'$ from $R$ with $\Sigma$, where $R' \subseteq R$, is the set of functional dependencies equivalent to the set of functional dependencies $X \to Y$ in $\Sigma^+$ such that $X \subseteq R'$ and $Y \subseteq R'$

Have to look at the closure

$X \to Y$
$X \subseteq R'$　$Y \subseteq R'$

## 4. Dependency Preserving

Definition

A decomposition of R with sigma into delta = {R_1, ..., R_n} with the respective sets of functional dependencies sigma_1, ..., sigma_n is **dependency preserving** if and only if

$$\Sigma^+ = (\Sigma_1 \cup \ldots \cup \Sigma_n)^+$$

****!!! Caution: in BCNF, a lossless join decomposition doesn't guarantee that it's dependency preserving!!!****
**In 3NF this can be guaranteed.**

## 5. BCNF Decomposition

Guarantees Lossless join　① Find any violations $X \to Y$
② Compute $X^+$ wrt $\Sigma$

**Algorithm**
BCNF Decomposition　violate the property　$X \to X^+ \supseteq Y$　$R_1$

**Algorithm #3: BCNF Decomposition**

Let $X \to Y$ be a functional dependency in $\Sigma$ that violates the BCNF definition *(i.e., non trivial and X is not a superkey)*. We use $X \to Y$ to **decompose** $R$ into two relations $R_1$ and $R_2$ in the following way:

$R_1 = X^+$
$R_2 = (R - X^+) \cup X$

$R_1 \cap R_2 = X$
$X \to X^+$ } lemma 1

We must now check whether $R_1$ and $R_2$ with the respective sets of **projected functional dependencies** $\Sigma_1$ and $\Sigma_2$ are in BCNF. If they are not, we **recursively** continue the decomposition.

# Lecture 10 Third Normal Form ===========================

## 1. Definition & Checking

**Normal Form**

- Based on the concept of transitive dependency.
- A relation that is in 1NF and 2NF and in which no non-key attribute is transitively dependent on the primary key.

**3NF**
Theorem

RHS is singular　BCNF doesn't care about it

**Third Normal Form**

A relation $R$ with a set of functional dependencies $\Sigma$ is in **3NF** if and only if for every functional dependency $X \to \{A\} \in \Sigma^+$:

- $X \to \{A\}$ is trivial, or
- $X$ is a superkey, or
- $A$ is a prime attribute

$\Rightarrow$ Negation Violation

$\exists$ FD $X \to \{A\} \in \Sigma^+$ s.t.
① $X \to \{A\}$ is non-trivial AND
② $X$ is not a Superkey AND
③ $\{A\}$ not in prime attribute

LEMMA 4. *A relation $R$ is 3NF iff for every elementary FD of $R$, say $X \to A$,*
*(a) $X$ is a key for $R$, or*
*(b) $A$ is a key attribute for $R$.*
PROOF. Easy.

**Note**

For relation $R$ **before decomposition**, it is sufficient only to look at $\Sigma$.

## 2. 3NF Decomposition (Bernstein's Algorithm)

**Normal Form**　Minimal Cover ($\Sigma'$)
$X \to Y$

**Algorithm**
Synthesis　$\Sigma$ ||

**Algorithm #5: 3NF Synthesis (Bernstein Algorithm)**

When a relation is not in 3NF, we can **synthesize** a schema in 3NF from a **minimal cover** of the set of functional dependencies.

- For each functional dependency $X \to Y$ in the minimal cover, create a relation
  $R_i = X \cup Y$
  Unless it already exists or is **subsumed** by another relation *(with some exceptions...)*.
- If none of the created relations contain one of the keys, pick **any** candidate key and create a relation with that candidate key.

"Synthesizing Third Normal Form relations from functional dependencies"

*We still call the synthesis method a decomposition because we decompose a relation into multiple relations.