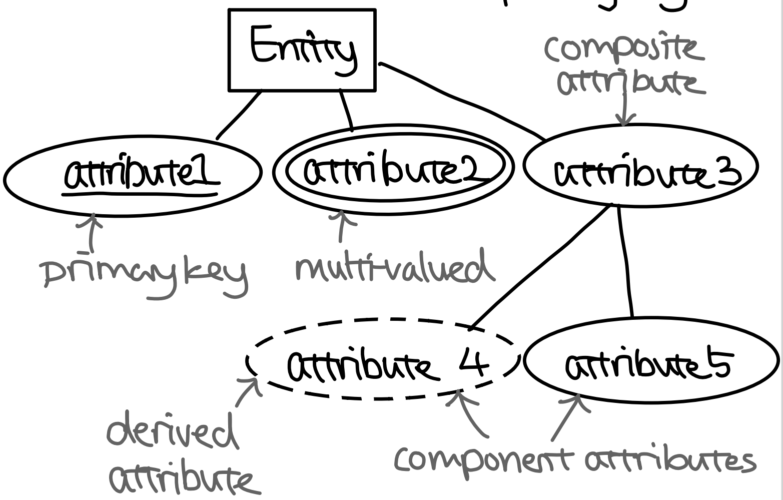
Chapter 2: ER model

- super key: a combination of keys that uniquely identifies a row
- candidate key: minimal super key
- primary key: one of the candidate key is chosen to be primary key



Weak entity set
→ must be a total & many to one relationship

Specialization & Generalization
- disjoint: must be either one (have to specify)
- overlapping: can be both (default)

Entity vs. Attributes
- many-to-one (attribute & entity)
- many-to-many (entity only)
- has separate attribute (entity only)
  ↳ Sometimes can be composite attribute

Entity vs. Relationship set
- has separate relationship with another entity (entity)

Relational tables
- composite attribute flattened out
  ↳ each component attribute act as a column
- multi-valued attribute has separate table
  ↳ with parent entity primary key as a column
- weak entity should have primary key of identifying entity set as a column
- many-to-many relationship should have a separate table with composite primary key
- many-to-one / one-to-many have separate table, with oneside primary key on manyside
- one-to-one relationship should have one side chosen to act as the many side
- specialization methods:
  1) lower-level entity set contains primary key of higher-level entity set (best)
  2) lower-level entity set contains inherited attributes
  3) when specialization is total, higher-entity set not require table

Foreign key
- constrains elements that can be added to table

Chapter 3: SQL

```
CREATE TABLE Table
(                                        ↙never negative
   table_key INT UNSIGNED,
   table_name VARCHAR(15) NOT NULL,
   PRIMARY KEY(table_key), ← constraint
   FOREIGN KEY(table2_id) REFERENCES
                              Table2(table2_id)
); ENGINE = INNODB;
                ↑
DROP TABLE Table    only to support foreign keys
ALTER TABLE Table ADD table_phone INT(12),
ALTER TABLE Table DROP table_phone;
ALTER TABLE Table ADD PRIMARY KEY(table_key);
```

- with foreign key constraint, cannot drop referenced table should drop constraint first

LIKE clause
- percent (%): any substring, no length limit
- underscore (_): any character, matches length

* different from '=', since '=' matches exactly, LIKE matches have flexibility

Join
- doesn't join NULL values
  ⇒ use OUTER JOIN

Order
SELECT FROM IN WHERE IN GROUP BY HAVING IN ORDER BY (ASC, DESC)

UNION is built-in DISTINCT OR
INTERSECT = table join
EXCEPT = NOT IN

Nested queries

IN to extract a column table_id IN(...)

SOME to compare a column from the outer query to a column in the inner query, return true even if one row in nested matches
return false if nothing in nested query
table_id > SOME(...)

ALL similar to SOME but all rows in nested query matches should match to return true
return true if nothing in nested query

EXISTS see if selected attribute row exist in the nested query
return false if nothing in nested query

Null - unknown value or does not exist
null arithmetic returns null
null comparison returns UNKNOWN
X IS UNKNOWN check if unknown
X IS NULL check if null

WHERE & HAVING returns false if UNKNOWN

aggregation ignores null values

View & Authorization

```
CREATE VIEW view_name AS (nested query)
GRANT SELECT/UPDATE(column) ON table_name
                              /view_name TO role/user
CREATE ROLE role_name
GRANT role_name TO user
```

Assertion

- ensures a certain condition will always exist
- assertion is checked everytime the involved tables are updated

```
CREATE ASSERTION assertion_name CHECK (
  (SELECT COUNT(*) FROM Table1) ≤
  (SELECT COUNT(*) FROM Table2)
);
```

Chapter 4 : Relational Algebra

Basic operators:	Additional operators:
select (σ)	set intersection ($\cap$)
project (π)	natural join ($\bowtie$)
union ($\cup$)	assignment ($\leftarrow$)
set difference ($-$)	left outer join (⟕)
Cartesian product ($\times$)	right outer join (⟖)
rename (ρ)	division ($\div$)

Select (σ) filters data (= WHERE)

Project (π) gets wanted columns (= SELECT)

Union ($\cup$) requires tables to have same # of attributes and attributes should be comparable (same type & same # of attributes) combines column

Set difference ($-$) requirements are same as Union

Cartesian product ($\times$) requires no attributes to have the same name

Rename (ρ) notation is $\rho_x(E)$ where table E is renamed to X

Set intersection ($\cap$) tables must have same # of attributes $R \cap S = R - (R - S)$

Assignment ($\leftarrow$) used to simplify the query

Outer join:

left: $R ⟕ S = (R \bowtie S) \cup (R - \pi_R(R \bowtie S) \times \{null, \ldots, null\}$

right: $R ⟖ S = (R \bowtie S) \cup (S - \pi_S(R \bowtie S) \times \{null, \ldots, null\}$

Division ($\div$)

let $S \subseteq R$ (S subset of R)

$R \div S = \{t \mid t \in \pi_{R-S}(R) \land (\forall s \in S, ((t \cup s) \in R)\}$

Aggregation ${}_G\mathcal{G}_{F(A)}(E)$

G: GROUP BY

F: function (i.e. SUM, COUNT, etc.)

A: attributes used for function (i.e. COUNT(A))

Chapter 5 · Functional Dependencies

Lossless-join decomposition: recover original table with natural join

Dependency preserving: if cannot be found, derive the dependencies (if candidate key on LHS, then automatically considered to be included

BCNF: decomposited table has no redundancy

↳ testing:

all LHS of non-trivial function should be a superkey of the table

BCNF $\not\Rightarrow$ Dependency preserving

Chapter 6: File & Storage

- data loaded to memory before access to disk

Magnetic disk

- each platter has 2 surfaces
- each surface has tracks
- tracks have sectors (typically 512 bytes)

surface > tracks > sectors (512 bytes)

Data block

- 4kB ~ 16kB (spans over sectors)

* should be a multiple of the sector size

Time for one I/O operation = seek time + rotational latency + transfer time

RAID 5: having parity bit in different disks

Buffer .

- buffer hit requires no I/O
- buffer miss requires I/O

* read & write both requires buffer

Access time

- seek time

maximum seek time = # tracks on one surface × time it takes for the head to move 1 track

- rotational latency: time required to rotate once

i.e. 10000rpm ⇒ $\frac{1min}{10000} \times \frac{60000ms}{1min} = 6ms$

- data transfer rate

block size ÷ sector size = x sectors

$360 \times (x \div$ # sectors in track$) = y$ degrees

$(y \div 360) \times$ rotational latency $= z$ transfer rate

Chapter 7: Indexing

- primary indexing: how the actual file is organized
- secondary indexing: used to search the file

B^+-tree

- leaf node: at least $\lceil \frac{(n-1)}{2} \rceil$, at most $(n-1)$ values
  n pointers
  last pointer used to chain leaf node
- non-leaf node: at least $\lceil \frac{n}{2} \rceil$, at most n pointers

Merging / splitting

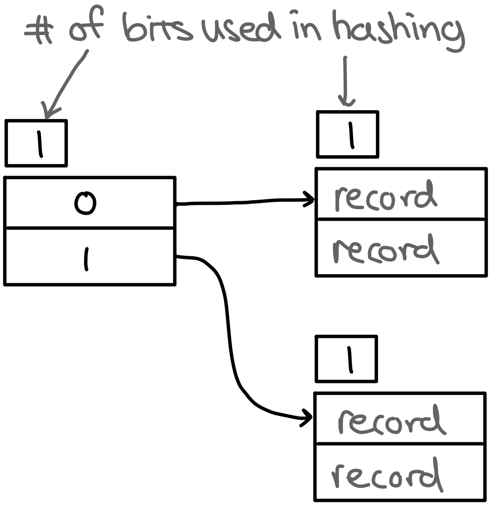
- redistribute keys

↳ if not merging / splitting, simply update the keys

- merge with sibling node

Hashing

- static hashing: hash function: k mod #
- extendable hashing:



if same value added constantly, chaining is used

* when maximum bits are used (cannot extend bucket address table)