DMDB - Summary

Relational Algebra Operators:
$R \times S$: Cartesian Product
$R \bowtie S$: Combine all elements in which all common attributes have same values, no duplicates
$R \rtimes S$: Same as $\bowtie$, but only keeps right attributes
$R \ltimes S$: Same as $\bowtie$, but only keeps left attributes
$R \bowtie_{\theta} S$: Does **cartesian product** $\times$ and then filters for condition θ
$R \bowtie_{A} S$: Same as $\bowtie$ but only matches column A (subclass of theta-join $\bowtie_{\theta}$)
$R - S$: Removes all rows, which are also in S
$R \div S$: Keeps rows (without the cols of S), of which there exists copies whose cover all attributes of S.
$R ⟖ S$: Same as $\bowtie$, but includes all rows from S and fills in missing values from R with nulls.
$R ⟕ S$: Same as $\bowtie$, but includes all rows from R and fills in missing values from S with nulls.

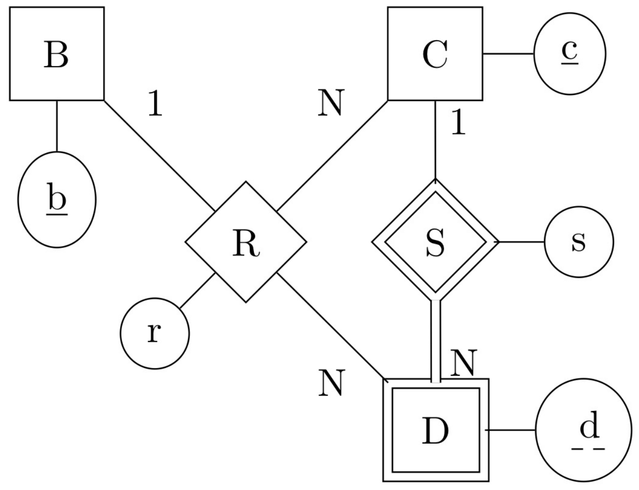
Candidate Key: Set of attributes, which can uniquely determine every element only using all attributes in the set. E.g. if one element can be removed from the set and it still uniquely determines every set, it's not a candidate key.
Primary Key: Set of attributes chosen as key (must be unique)
Secondary Key: Set of attributes not chosen as key

Converting ER-Diagrams to Relational Model:

1. Write down all *entities* and their attributes with keys as described.
   Special Cases:
   - Weak Entities: Include all attributes of the relation (except the relations attribute) and set all of them to the key.
   - Generalisation: Use the referenced entity's key as key for the referencing entity
     E.g. $B(\underline{b}), C(\underline{c}), D(\underline{c, d})$
2. Write down all *relations* and their keys as "attributes" and also their own attribute.
   If there's a conflict, e.g. a Weak Entity and thus we would end up with c twice, rename it to c' and include both c and c'.
   As for the keys, for each Entity with a relationship marked with 1, set all other elements (except relation attribute) as key. If there are multiple options, list them as extra keys.
   E.g. $R(b, \underline{c, c', d}, r), S(\underline{c, d}, s)$
3. *Merge relation* with entity A iff there exists another entity in the relation marked with 1 and A contains all keys of all other entities except the one marked with 1.

E.g. $B(\underline{b}), C(\underline{c}), D(\underline{c}, d, s), R(b, \underline{c, c'}, d, r)$

Example used:



ER model 3

Normal Forms:

- *1NF*: No divisible attributes, e.g. no lists as attributes.
- *2NF*: There exists no RHS attribute, which is *not part a key* that depends *on only part* of a candidate key.
- *3NF*: Each non-trivial FD either has a superset of a candidate key as LHS or the RHS contains part of a candidate key.
- *BCNF*: Each non-trivial FD has a superset of a candidate key as LHS.
- *4NF*: Each non-trivial MVD has a superset of a candidate key as LHS.

3NF Synthesis: Lossless, keeps FDs but can retain redundancies

1. Compute the minimal basis F_c
2. For all $\alpha \to \beta \in F_c$, create a new relation with $\alpha \cap \beta$, e.g. $R_{\alpha \cup \beta}(\alpha \cup \beta)$
3. If none of the above relations contains a superkey, add a relation with a key
4. Eliminate a relation, if it's subset of another relation.

BCNF (and 4NF) Decomposition: Lossless, removes redundancies, but can lose FDs

1. Find all evil FDs (MVDs), usually they are given, but note that there are two approaches.
   Either one also considers the trivial FDs (MVDs), e.g. if $A \to B$ and $B \to C$ one also considers $A \to C$ as an FD, or one calculates the closure for every FD, e.g. in the previous example one would get $A \to BC$ and $A \to C$. The latter is how it's explained in FS23.
2. Let $\alpha \to \beta$ ($\alpha \to\to \beta$) be the evil FD (MVD) present inside a certain relation
   We will proceed to split this relation into two relations.

3. First part is a relation with α and β, e.g. $\mathcal{R}_i^1 = \alpha \cup \beta$
4. Second part is a relation with all remaining elements of that relation except β, e.g. $\mathcal{R}_i^2 = \mathcal{R}_i - \beta$
5. Now that we have split the table, we check again in all tables if there's an evil FD (MVD), e.g. if any evil $\alpha \rightarrow \beta$ ($\alpha \rightarrow\rightarrow \beta$) is still in a relation and $\alpha \cup \beta$ is not the set of attributes of the whole relation (e.g. α is a key and thus the relation is not evil anymore).
   If so, we go back to (1.) otherwise we are done.

MVD

Intuitively: If we have $A \rightarrow\rightarrow B$, then if we fix A we get a set of values for B (let's call it A_B) and a set of values for C (any set of remaining attributes, call it A_C). If we now fix for any value in A_B, we get a set of values X of attributes C. It must hold, that for any other value we fix in A_B, we also get this same set of values X. The same must also hold if we fix a value in A_C (with X as a set of values of B).

A	B	C
a	b	c
a	bb	cc
a	bb	c
a	b	cc

Table: Example from class, with $A \rightarrow\rightarrow B$ (and thus also $A \rightarrow\rightarrow C$, see rules below).

MVD Rules:
Trivial MVDs:

- $\beta \subseteq \alpha \Rightarrow \alpha \rightarrow\rightarrow \beta$
- $\alpha \rightarrow \beta \Rightarrow \alpha \rightarrow\rightarrow \beta$ (e.g. if $R \subseteq \alpha \cup \beta$)
  Complement:
- $\alpha \rightarrow\rightarrow \beta \Rightarrow \alpha \rightarrow \mathcal{R} - \alpha - \beta$
  Multi-value Augmentation:
- $\alpha \rightarrow\rightarrow \beta \wedge (\delta \subseteq \gamma) \Rightarrow \alpha\gamma \rightarrow\rightarrow \beta\delta$
  Multi-value Transitivity:
- $(\alpha \rightarrow\rightarrow \beta) \wedge (\beta \rightarrow\rightarrow \gamma) \Rightarrow \alpha \rightarrow\rightarrow \gamma$

Does a tuple have to exist?

1. For each MVD $\alpha \rightarrow\rightarrow \beta$:
2. If we fix α and β to the given values in the tuple we get set S of possibilities for the remaining attributes.
3. Now we search for any other remaining attribute values not in S if we fix another α or another β. These remaining attribute values are then "missing" because of the MVD.

4. If one of these "missing" values is given in the tuple (for fix α and β), the tuple is necessary.
5. If for all MVDs there is no such missing value provided in the tuple, it is not necessary.

Recoverability Classes:

1. **Recoverable (RC)**: If T_1 *reads* a value written by T_2, T_2 has to *commit first*.
2. **Avoids cascading aborts (ACA)**: If T_1 *reads* a value written by T_2, T_2 has to be *committed at the point* where T_1 *reads*.
3. **Strict (ST)**: If T_1 *reads/writes* a value written by T_2, T_2 has to be *committed at the point* where T_1 *reads/writes*.

Locking and Snapshot Isolation:

- *Two-Phase Locking (2PL)*: A thread has two phases, first phase only acquire, second only release. (Two types of lock states, shared and exclusive for read and write respectively)
- *Strict Two-Phase Locking (S2PL)*: A thread acquires locks during the transaction and keeps the lock until it commits. (Two types of lock states, shared and exclusive for read and write respectively)
- *Snapshot Isolation (SI)*: Just like git. Read old version until committed. If someone committed after I started, which has *written to the same object*, abort.
- *Phantom Read*: Happens when a transaction reads a set of rows that satisfy a condition, another transaction inserts/deletes/updates rows, and the first transaction re-reads and gets a different set of rows.

Serialisability:

- *Serialisable*: Equivalent to serial history
- *View Serialisable*: Same as serialisable, but additional constraint that the first read on each object is the same in both histories.
- *Conflict Serialisable*: Equivalent to a serial history, by swapping non-conflicting operations (e.g. any combination of two different thread operations on the same object involving a write operation).
- In terms of *strength*, it holds that $\text{Conflict Serializability} \subseteq \text{View Serializability} \subseteq \text{Serializability}$

Query Optimisation:

- *OLAP* (Analysis of large data): Column Storage, Iterator Model
- *OLTP* (Transactions, smaller data): Row Storage, Materialisation Model
- *Iterator Model*: Iterates over operators using `next()`
- *Materialisation Model*: Processes input all at once and then directly outputs

- *Logical Plan*: An abstract representation of what operations need to be performed to execute a query, focusing on the *"what"* rather than the "how."
- *Physical Plan*: A detailed plan that specifies how the logical operations will be physically executed by the database, focusing on the *"how."*

Calculating the number of IOs, first figure out how many tuples fit into the two buffers (usually one page per relation), then choose this as block size. Subsequently $B(R) = \frac{|R|}{\text{block\_size}}$.
Note that the following might be wrong if one of the relations fully fit into one of the buffers and no inner or outer relation is specified.

- *Nested Loop Join*: $B(R) + |R| \cdot B(S)$ IOs
- *Block Nested Loop Join*: $B(R) + B(R) \cdot B(S)$ IOs
- *Index Nested Loop Join* (equi-join): $B(R) + |R| \cdot C$ IOs
- *Naive Hash Join*: $B(R) + B(S)$ IOs
- *Sort Merge Join*: $B(R) + B(S)$ IOs
- *Grace Hash Join*: $3 \cdot (B(R) + B(S))$ IOs