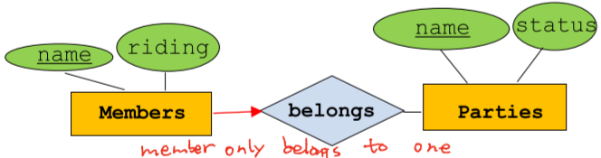
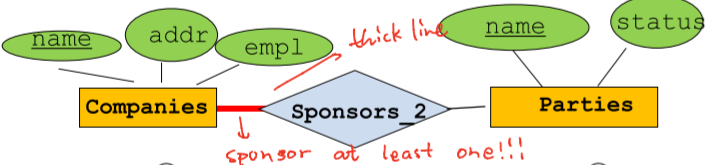
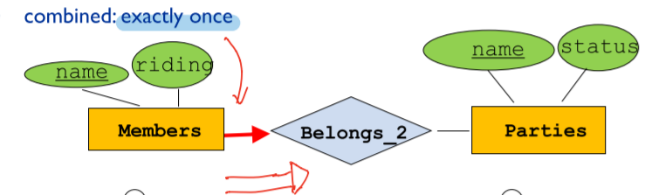
**COMP 421 cheat sheet (Yingjie Xu)**

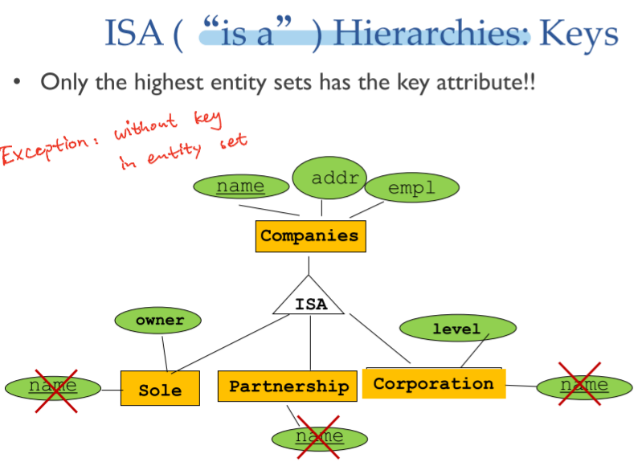
**Entity-Relationship Model (ER):** representation of the data model of the application

**Entity (instance in OOP):** an entity is described using a set of **attributes.**

**Entity Set (class in OOP) 🡪 rectangle in ER:** All entities in an **entity set** **🡪** **(oval in ER)** have the same attributes. (An entity set must have a **Key** 🡪 underlined)

****Entity 🡪 Rows; attributes 🡪 Cols

**ISA (“is a”) Hierarchies 🡪 Subclasses:** A ISA B, then every A entity is also a B entity – Key only in B.

**Reason** for ISA: 1. Additional descriptive attributes specific for a subclass 2. Identification of a subclass that participates in a relationship.

**Overlap Constraint:** Can an entity be in more than one subclass? (allowed/disallowed)

**Covering Constraint:** Must every entity of the superclass be in one of the subclasses? (yes/no)

**Relationship**: Association among two or more entities.

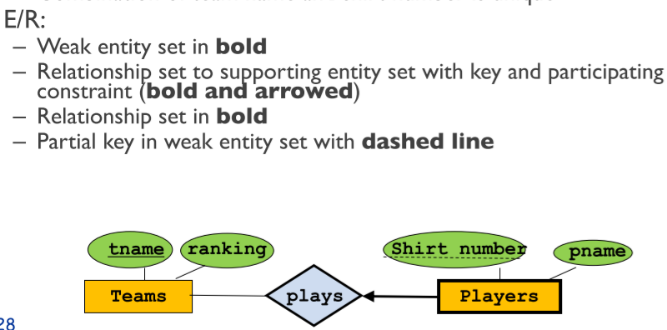
**Relationship Set**: Collection of similar relationships.

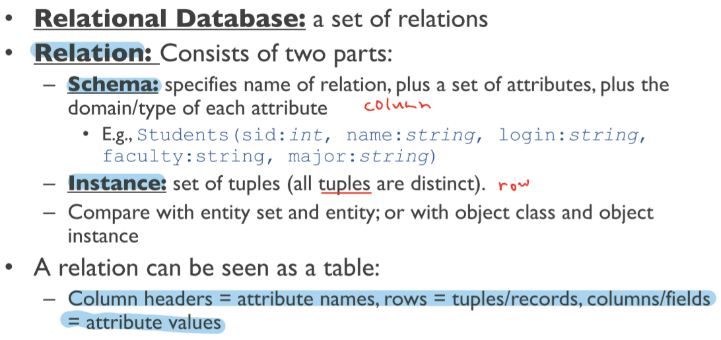
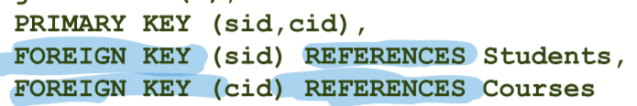
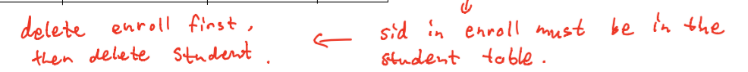
**Many-to-Many**: …

**Key-constraints (one-many, many-one):** we must have **at most** one entity set is participating in the relationship.

**Participation constraints:** we must have **at least** one entity set is participating in the relationship.

**Ternary Relationship** is relationships involving 3 entity sets. Keep in mind that a ternary relationship database entry MUST include all 3 entities.

**Weak Entity:** a weak entity can be identified uniquely only by considering the primary key of another (owner) entity.

Don’t keep redundant information.

Schema = column header + table name

Column header = attribute names

Row = tuple / record

Columns / fields = attribute values

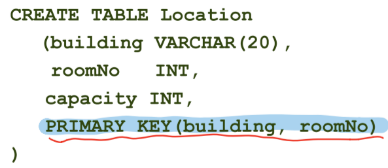
All rows are distinct in DB

Database Schema: collection of relation schemas

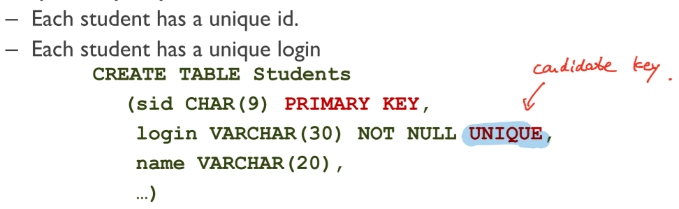
**Data Definition Language (DDL):** defines the schema of a database.

**Data Manipulation Language (DML):** manipulates the data

**Integrity Constraints** must be true for any instance of the database

**- Not Null:** requires an attribute to always have a proper value

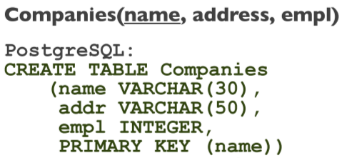
- **Primary Key Constraints:** No two distinct tuples can have same values in all key fields. (**unique**) PK should be **not null**.

- **Candidate Key Constraints: (UNIQUE)**

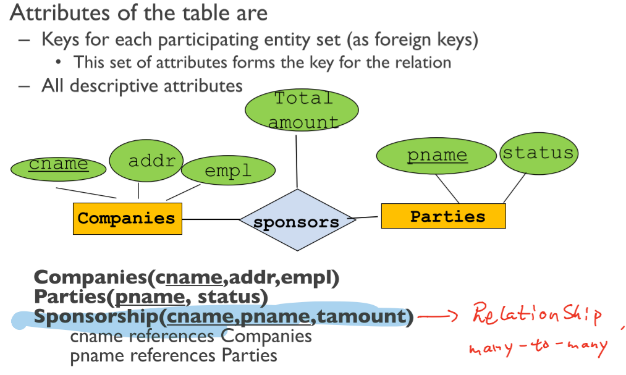
**- Foreign Key Constraint:** Set of attributes in one relation R that is used to “refer” to a tuple in another relation Q. the foreign key value of a tuple must represent an existing tuple in the referred relation.

If all foreign key constraints are enforced, **referential integrity** is achieved, i.e., no

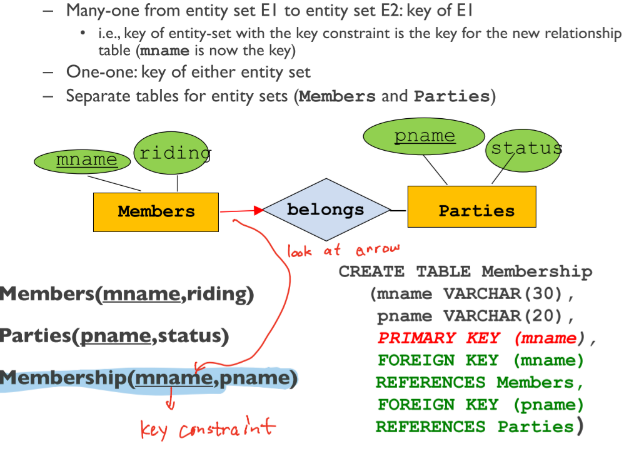
dangling references.

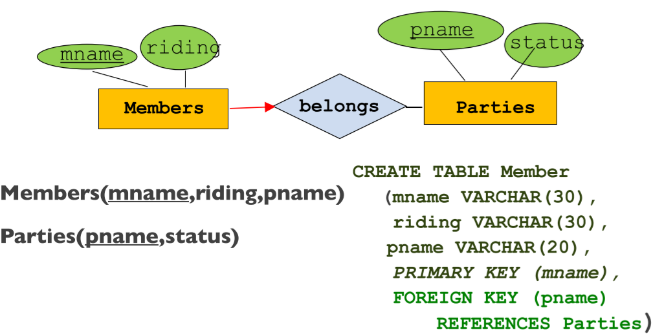
**ER-Relational Translation**

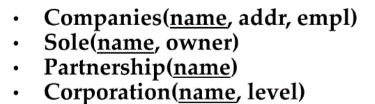
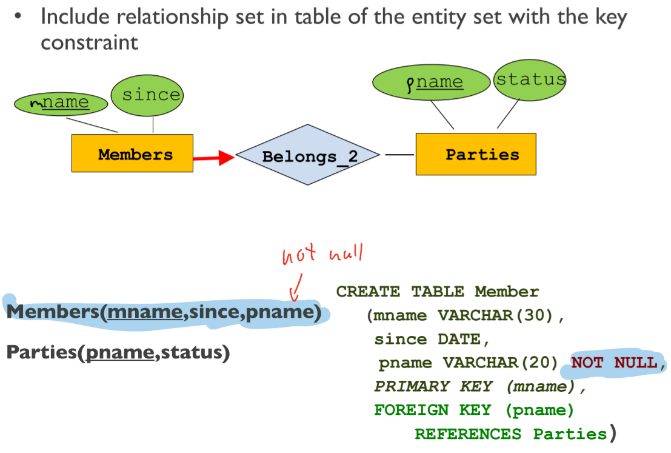
**- Entity Sets to Relations:**

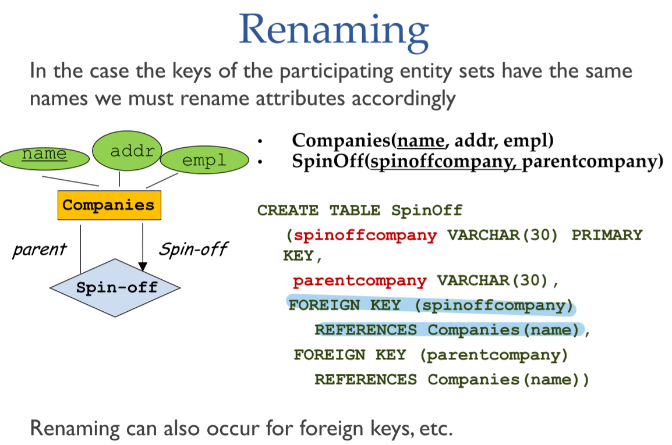
**- Many-many Relationship Sets:** A many-to-many relationship set is **ALWAYS** translated as an individual table.

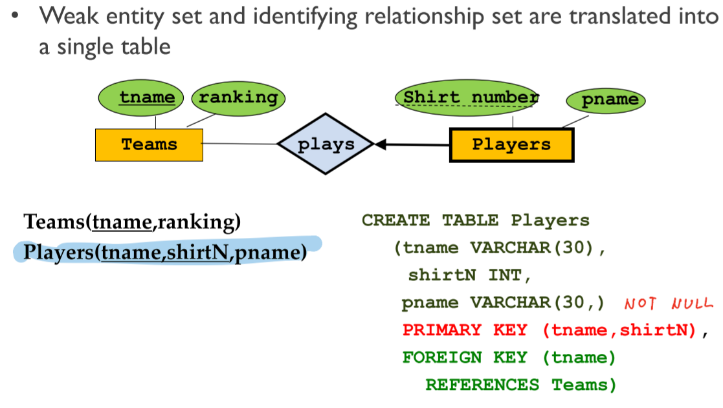
- **Relationships Sets with Key Constraints:**

** - Alternative 1:** map relationship set to table

 **- Alternative 2:** include relationship set in table of the entity set with the key constraint

**- Key and Participation Constraints**

**- Participation Constraints:** cannot be reflected usually, except for both key and participation.

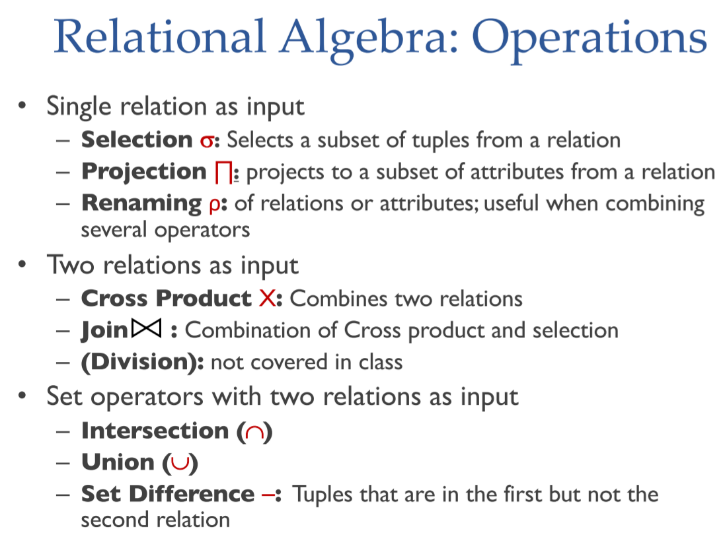
****- **Weak Entity Sets**

**- Translating ISA Hierarchies**

 - General Approach: distribute information among relations. Relation of superclass stores the general attributes and defines key. Relations of subclasses have key of superclass and additional attributes.

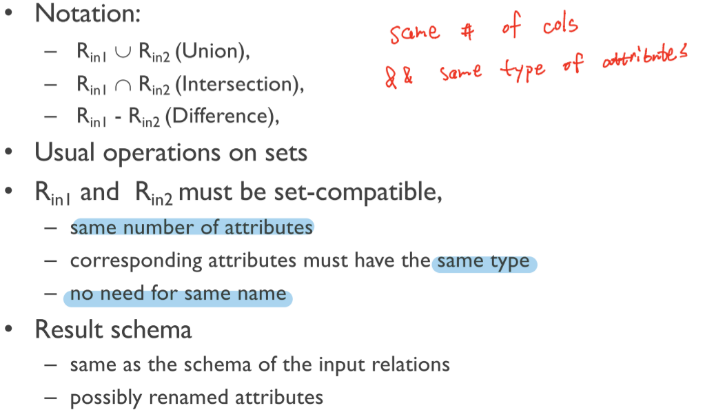
- Object-oriented approach: Sub-classes have all attributes; if an entity is in a sub-class it does not appear in the super-class relation.

- One big relation: Create only one relation for the root entity set with all attributes found anywhere in its network of subclasses. Put NULL in attributes not relevant to a given entity.

**Relational Algebra**

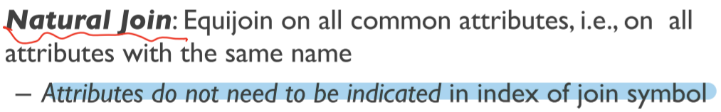
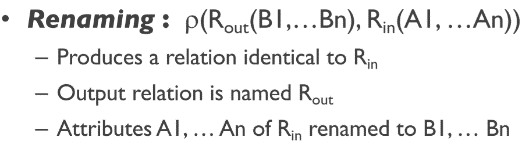
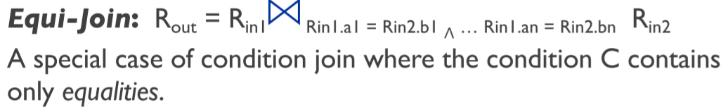
***Output will not have any duplicated results***

***Relational algebra doesn’t care about keys***

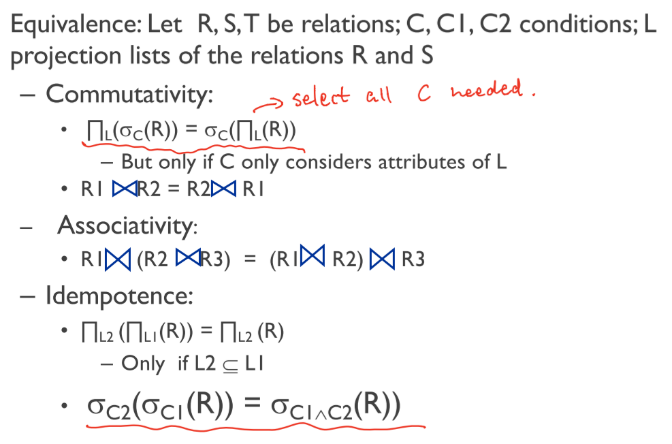


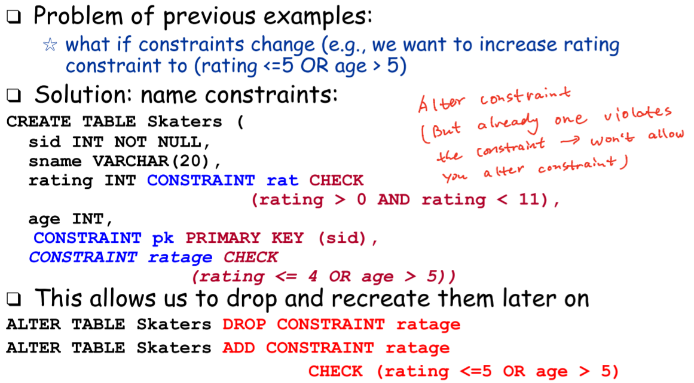
A – B: everything in A and not in B

**Cross-Product:** Each row of first table is paired with each row in second table|A| X |B| = |A X B|

******Joins = cross-product + selection**

****

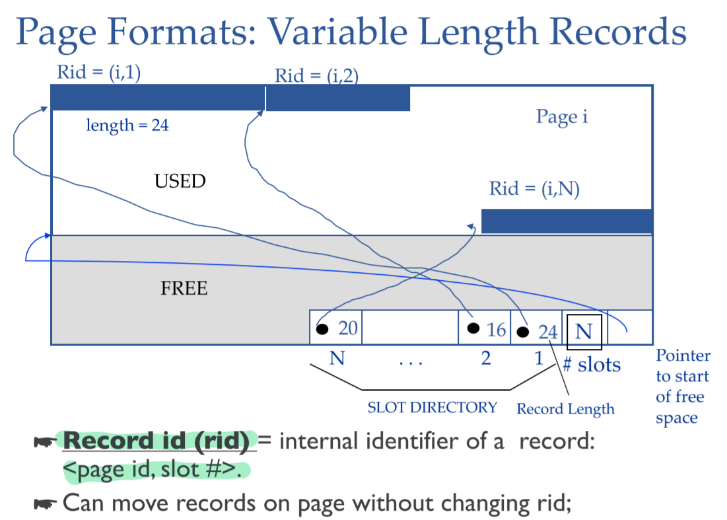
**Rules:**

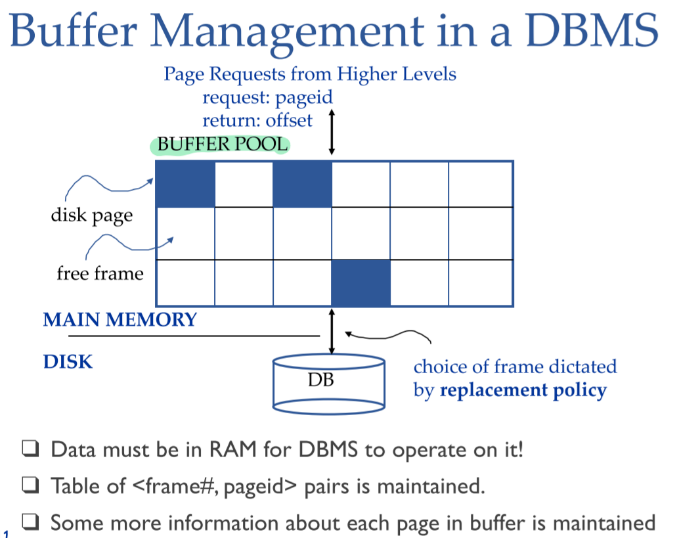
**Integrity Constraints (CHECK)**

**Internal of DB**

Query optimization and execution 🡪 relational operators 🡪 file and access methods 🡪 buffer management 🡪 disk space management

**Structure**: register 🡪 cache 🡪 main memory 🡪 disk

**block = unit of transfer for disk r/w = page = frame**

To change a page, bring the page from the disk to memory and then change.

Buffer pool stores <frame #, page id>

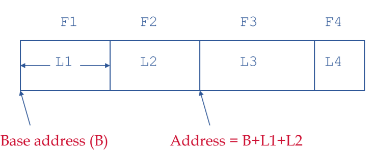
**Buffer manager** will load up data to buffer pool when upper layer sends a request.

**Loading a page from disk:**

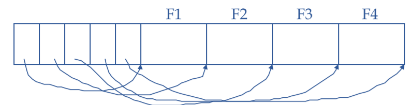
If requested page is not in pool: If there is an empty frame 🡪 Choose empty frame.

Else (no empty frame) 🡪 Choose a frame for replacement 🡪 If frame is **dirty** (page was modified), write it to disk 🡪 Read requested page into chosen frame.

**Page Pin**: replacement frame has pin counter 0. When requesting a page that is in the buffer 🡪 increment the pin counter. After finishing the operation 🡪 decrement pin counter (set dirty bit if page has been modified). Replacement policy 🡪 only pin counter = 0 could be chosen to replace.

**Record Format**

1. Fixed length (Works with fixed-length types)

2. Variable length (efficient storage of nulls)

**(record id) Rid = <page id, slot #>**

**File = collection of pages**

**Unordered (Heap) File:** Suitable when typical access is a full scan of all records

**Sorted file:** Records are sorted by one of the attributes (e.g., name).

**Indexes:** We call the collection of attributes over which the index is built the ***search key* attributes** for the index.

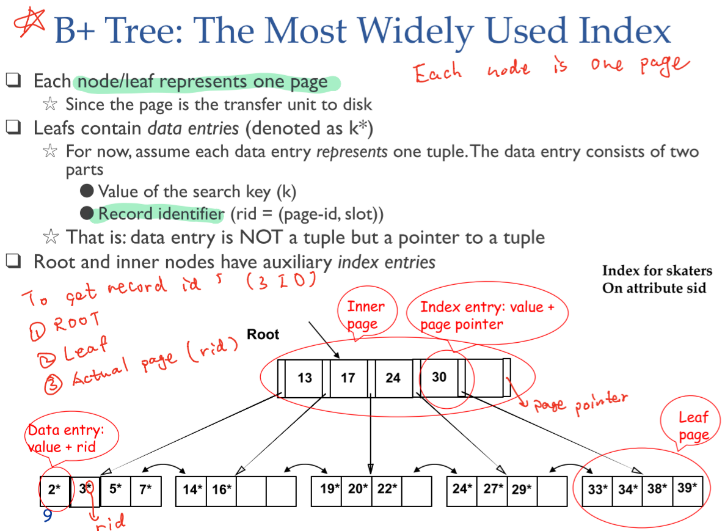
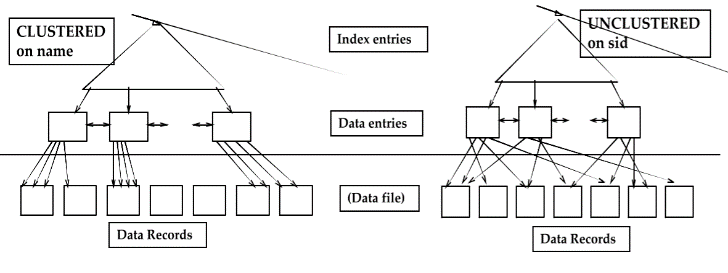
**indirect Indexing:**

**- Indirect Indexing I:** on non-primary key search key: (2015, rid1), (2015, rid2), (2015, rid3), … 🡪 several entries with the same search key side by side

**- Indirect indexing II:** on non-primary key search key: (2015, (rid1, rid2, rid3,…))

**Direct Indexing:** store record directly instead of rid

*Primary vs. secondary:* If **search key** contains primary key, then called primary index. (unique)

*****Clustered vs. unclustered:* Def of **clustered**: Relation in file sorted by the search key attributes of the index. (A file can be clustered on **at most one** search key.) Cost of retrieving data records through index varies greatly based on whether index is clustered or not!

**Node = Page; Inner page + Leaf page**

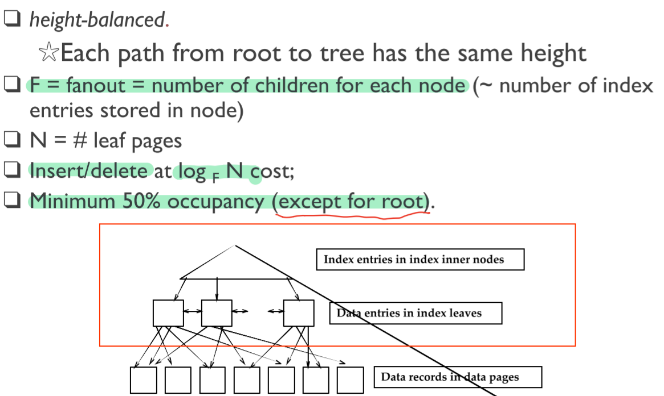
**Data Entry = Leaf = value of search key + rid**

**Index Entry = page pointer (point to the next level of B+ tree) + value = Root + Inner nodes (min 50% occupancy)**

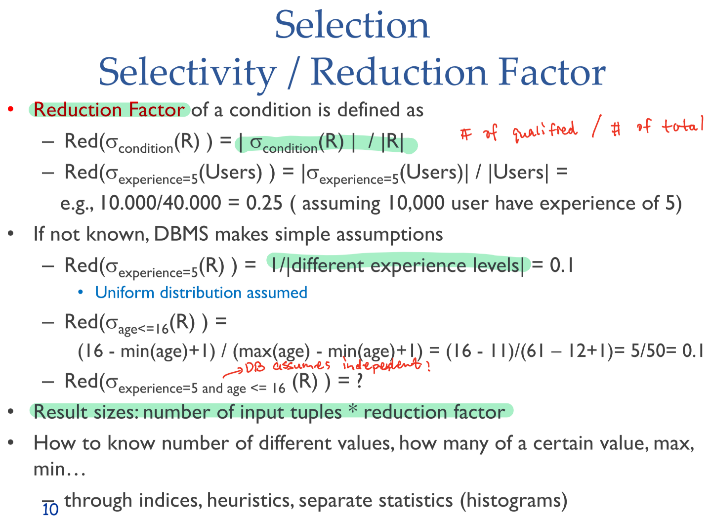
**Height-balanced**

**Fanout = F = number of children for each node**

**N = number of leaf pages**

**Cost of insert/delete = logF(N)**

**Cost Model for Execution: number of I/Os**

assumption that the root and all intermediate nodes of the B+ are in main memory: only leaf pages may not be in main memory!

Indices usually only useful with very **small reduction factors**

