Types of Dependencies

- **Into-key dependency**: A key in one table that matches the PK of another table.
- **Transitive dependency**: A key that is a superset to the PK.
- **Partial dependency**: A key used for locating records.
- **Superkey**: A key that can uniquely identify each row in a table.
- **Candidate Key**: A key used to physically sort the stored data.
- **Primary Key**: A key that is a candidate key for doing that.
- **Foreign Key**: A key that is chosen in one table and is used to locate records in the other table.

Minimal Super Key = Candidate Key: A key that can uniquely identify each row in a table.

**Primary Key**: The chosen Candidate Key for doing that.

**Secondary key / Alternate key**: A Candidate Key not chosen for doing that.

**Search Key**: A key used for locating records.
**Sort or control key**: A key used to physically sort the stored data.

**Composite key or concatenation key**: A key with >1 columns. (Usually implies "composite primary key").

### Types of Dependencies

**From the full key**: full PK -> outside of the PK

**Partial dependency**: part of the PK -> outside of the PK

**Into-key dependency**: outside -> into the PK

**Are candidates**

- **First Normal Form (1NF)**: there are fixed numbers of columns.
- **Second Normal Form (2NF)**: 1NF and no partial dependencies.
- **Third Normal Form (3NF)**: 2NF and no transitive dependencies.

**Simplified Armstrong's Axioms**

- Reflexivity: X → X
- Transitivity: X → Y  Y → Z  X → Z
- Union: X → Y ↔ X → Y or X → Z
- Decomposition: X → Y Z ↔ X → Y X → Z
- Pseudotrans.: X Y → W Y Z ↔ W X Z

**Conflicts**

- If it can be transformed into a serial schedule by swapping non-conflicting operations, Two operations are said to be conflicting if all conditions satisfy:
  1. They belong to different transactions or have to acquire locks on the same data item involved in a specific order.
  2. They operation on the same data item at the same time.
  3. They can not happen in parallel.

**Deadlocks**

- If multiple transactions acquire locks on data items in a specific order, a transaction does not have to acquire locks on each data item involved, then no cycle can happen.

**Update, Insert, Delete Commands and Useful Funcs**

- **JOIN**: Merge Sort Join, Merge Cost: (rR) + (sS) + Sort Cost
- **Block Nested Loop Join**
- **Index Nested Loop Join**

**SQL Queries**

```
INSERT INTO table_name (Column1, Column2, ...) VALUES ('Value1', 'Value2', ...);
DELETE FROM table_name WHERE column = some_value;
UPDATE movies SET invoice = 'paid' WHERE paid = 0;
CREATE table Person (attribute1 ...
CREATE table BookCopy( ...);
```

### Definitions

- **Superkey**: A key that contains >1 columns.
- **Candidate Key**: A key that can uniquely identify each row in a table.
- **Primary Key**: A key chosen in one table for doing that.
- **Foreign Key**: A key that is chosen in one table and is used to locate records in the other table.

**Superkey**: A key that contains >1 columns.

**Candidate Key**: A key that can uniquely identify each row in a table.

**Primary Key**: A key chosen in one table for doing that.

**Foreign Key**: A key that is chosen in one table and is used to locate records in the other table.
Key Constraint: e.g. Any [Department] can only <be managed> by 0 or 1 [employee]; [Dep.] <manage> [Emp.]
Total/Partial Participation: e.g. ● Not all employees get to manage — [Employees] partially part in <manage>; thin/single line.
ER Features: ○ Each department must be managed — [Departments] totally participate in <manage>; thick/double line.
Weak Entities: One-to-many relationship + Total Participation. Each Weak Entity can only have One Owner Entity!
○ One-to-many relationship — The identifying relationship of the weak entity: [The Owner] <--[The Weak]
Total Participation: All Weak Entities must all "totally" participate in the identifying relations
Class Hierarchies: Represented by a triangle with text "ISA": e.g. There are 2 types of [Users]: [Free Users] and [Premium Users].
Aggregation: Represented by a dashed box surrounding a collection of entities + relationships. E.g.: ○ [Employees]--->[monitors]--->[Projects]--->[sponsored_by]--->[Departments] | • A process contains these info:
  - Zero, 1, or more processes can monitor the process.
  - An employee can monitor 0, 1 or more processes.
  - A department does not necessarily have to sponsor any process.

Model for Accessing Costs and I/O Costs
p(I): # of data pages in table T r(T): # of records in table T D: average time to read/write a page
Seek Time: Cost switching tracks. Rotational Delay: Cost to go to a sector when head is on right track.
Page Transfer: Cost reading a page

db.orders.mapReduce( () => { (for [item in this.items] { // map emit(item.name, {count: 1, qty: item.qty}) }); }, (key, obj) => { // reduce count: obj.reduce( (sum, obj) => sum + obj.count, 0), qty : obj.reduce( (sum, obj) => sum + obj.qty, 0), } }, {out: {name: 'result1', query: {Key: 'val'}, finalize: finalizer } }, {key, res} => {res.avg = res.qty/res.count; return res });

Use Agg. Funcs. in FUNCTION: For each driver for whom there are > 2 bars serving some beer they like, print the tot. # of beers they like that are served by some SELECT 1.drinker, COUNT(DISTINCT 1.beer) AS total FROM likes 1 inner join serves s ON 1.beer = s.beer GROUP BY 1.drinker HAVING Count(DISTINCT s.bar) > 2

Print all airports that have direct flights to all destinations that PHL does.
Lv. 0: Flights departing from a F1 city and arriving in a F2 city.
Lv. 1: Destinations of PHL flights, where no flight from a F1 city will arrive.
Lv. 1: Cities that do not arrive in a city described in Query Lv. 2.

db.people.insert(name: 'Li'));
db.people.deleteMany({major: 'DAYS'});
db.people.deleteOne({name: 'Mary'});
db.old_Collection_Name_people.drop();
db.people.updateOne({id: 1}, {($set: {role: 'admin'})});
db.people.updateMany({id: 'one'}, {($set: {role: 'admin'})});
db.people.findOne({id: 'two'});
db.people.findOne({$pretty});
db.people.find({name: 'Li'});
db.people.insert({name: 'Li'});

SELECT DISTINCT depart FROM flight
WHERE NOT EXISTS
- OR FROM flight f2 WHERE f1.depart = f2.arrive
DEPART AND F2.arrive = 'PHL'

Print All Cities That Have No Direct Flight to PHL

SELECT depart FROM flight
WHERE NOT EXISTS
- OR FROM flight f2 WHERE f1.depart = f2.arrive
AND f2.arrive = 'PHL'

---

MongoDB Neo4J
MATCH (m):{*1..5}-(n:Person {name: 'Alice'})-[:LOVES {since: 2010}]-(p:Mention)
OPTIONAL MATCH p: (n)-[*..5]-(m) -- Assign a path to p.
Optional pattern: roles will be used for missing parts.
WHERE n.name = 'Alice' AND n.age > 10
WHERE (n.matchCount) AS numFriends
ORDER BY n.property DESC
RETURN n AS cnt, n.name AS name, n.age AS age

CREATE (n:Person {name: 'value'});
SET n:Spouse:Parent:Employee;

Neo4J Built-in Roles

DBMS Security

Bell-LaPadula Model: Top Secret (TS) > Secret (S) > Confidential (C) > Unclassified (U)
- Subject S can READ object O iff class(S)=class(O)
- Subject S can WRITE object O iff class(S)<=class(O)

Approaches to DBMS Security: (1) discretionary and (2) mandatory access control.

Privileges Reader Publisher Architect Admin
Change owner password
Read data
Termiate own query
Write/update/delete data
Manage index/constraints

Cost Estimates For Single-Relation Plans

Sequential scan of file: NPages(R) = Index I on primary key matches selection. Cost is Height + B-tree, i.e. about 1/log2(high) for hash index. Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.

Factors of matching select.