Data anomaly: inconsistency is DB as result of an update, insertion, and/or deletion
Logical independence: data is independent of the table structure, i.e. unaffected when a col is added or a col is moved
Physical independence: the data is independent from its physical container and unaffected by changes
Functional dependencies: partial and transitive dependencies
Locking: controls access that users have to the DB
Exclusivity: maintain data consistency when same data updated simultaneously by multiple transactions
Granularity: defines how fine the control of the lock is
Lost updates: 2 diff transactions try to update same col on same row within DB at same time, result of one is lost
Uncommitted data: 1 transaction updates data, but hasn't committed it permanently to the DB, b/c failure, transaction is rolled back and data item is returned to previous value
Inconsistent retrievals: transaction calculates summary function over set of data while other transactions update data
Concurrency: ability to allow multiple users to affect multiple transactions within a DB (multiple users use DB at once)
Concurrency control: manages simultaneous access to DB, prevents 2 users from editing same record at same time
Transaction: logical unit independently executed for data retrieval or updates, group of tasks (like a query for ex)
Transaction logfil: records all transactions and DB modifications made by each transaction (good for troubleshooting)
Snapshots: backup of DB
ACID: Atomicity, Consistency, Isolation, Durability
Atomicity: transaction treated as atomic unit, either all operations executed or none, no partial transactions
DB Consistency: DB remain in consistent state after any transaction, transaction should not have effect on data
Isolation: all transactions being executed simultaneously are treated independently
Durability: durable enough to hold latest updates even if system fails or restarts, committed transactions not lost
COMMIT: permanently save changes done in transaction in tables/DBs. DB cannot regain previous state after this
ROLLBACK: undo transactions that have not been saved in DB, undoes all changes since last COMMIT

Performance considerations for locking, exclusivity, granularity:

More difficult to access all of the data but safer, lower performance the more restrictive it is

Lock Granularities for accessibility and performance cost

DB: 1 person controls all data in DB
Table: select tables can be locked/accessed
Row: can only access certain rows
Column: can only access certain cells of all rows
As you go down, requires more resources and is harder to implement

SQL

Query Syntax: *select* vals *from* table1 t1 *join* table2 t2 *on* t1.id = t2.id *where* conditions *order by* val *asc/desc*;
Linking Table: *insert into* linkingTable (varNames) *select* id1, id2 *from* table1 *join* table2 *on* … *where* …; (make combos of keys)
Create: *create table* tableName (varName varType *primary key auto_increment*, varName2 varType2 *not null*, …);
Create Combo Key: create table w/o PK, then add: *primary key*(pkVal1, pkVal2)
Foreign Key: add val as normal val, then add *foreign key* (valName) *references* referenceTable(referenceKey)
Insert: *insert into* tableName (fieldsInserted) *values* (valsInOrder);
Or: *insert into* tableName (fieldsInserted) *select* val1, val2, … *from* tables; (instead of values… , do a query)

Codd's Rules: **1**) all info in RDB is logically represented as col vals in rows in tables. **2**) all vals in table must be accessible thru combo of table name, PK val, col name. **3**) nulls represented and treated in systematic way, indep of data type. **4**) metadata stored and managed as ordinary data, in tables in DB, data is available to allowed users using standard DB lang. **5**) RDB can have many langs, but must support 1 well defined declarative lang w/ data defn, view defn, data manip, integrity constraints, authorization, transaction management. **6**) any view that is theoretically updatable must be updatable thru the system. **7**) DB supports set level insert, update, delete. **8**) app programs/ad hoc facilities logically unaffected when physical access methods or storage structures are changed. **9**) app progs and ad hoc facilities logically unaffected when change made to table structures that keep original table vals (change order of cols or insert col). **10**) relational integrity constraints must be definable in relational lang & stored in system catalog, not at app level. **11**) end user and app progs unaware and unaffected by data location. **12**) if system supports low level access to data, must not be way to bypass integrity rules of DB.