Data management: used to capture, store, retrieve, analyze, present and interpret (large amount of) data

**Database**: an organized collection of related data, usually stored on disk [Data abstraction: abstract view of data]

**Database Management System (DBMS)**: a software system designed to store and manage databases

**Entity-relationship (ER) diagram**: a graph represents the main entities and corresponding relationships

* **Entity**: an object/ entity having a set of attributes (Entity set: the collection of all entities)
* **Attributes**: each attribute has a domain & each entity set has a key attribute
* **Relationship** (relationship degree: the number of participating entity types e.g. Binary R’, Ternary R’, N-ary R’)
* **Weak entities**: identified uniquely only by considering PK of another (owner) entity
* **Specialization** (subclass – has own specific attributes/R’ship) **& generalization** (superclass) [Extended ER (EER)]

Data model: collection of concepts that can be used to describe structure of database to achieve data abstraction

**Schemas**: metadata or data describing data – static & **Instance**: data in the database at particular time – dynamic

**Relational model**

Relations (R): a set of records (like table BUT not have merge columns/rows) Domains (D): a set of atomic values Attributes (A): name of role played by some domain in relation (degree of R: number of attributes in relation)

Tuples (T): ordered list of n values (relations are sets of tuples)

**Three-schema architecture**

* External level – provides access to particular parts of the database to users
* Conceptual/logical level – structure of a whole database for a community of users
* Internal level – the physical storage structure of the database

**Data independence**: ability to change the schema at one level of a database system without having to change the schema at the next higher level

**Integrity constraints (ICs)**: conditions that must true for any instance in DB (DBMS shouldn’t allow illegal instances)

* ICs are specified when schema is defined - ICs are checked when relations are modified

Types of ICs: [sequence to check violation of ICs: primary key (Entity, Key), Domain, Referential]

* **Domain constraints**: each attribute in a relation must belong to some domain (right stuff is in the right spot)
* **Key constraints (uniqueness constraint)**:
* **Entity constraints**: no PK can be null · **Referential integrity constraints** (data exists in ‘parent’ before ‘child’)
* **User-defined constraints**: general user defined constraints (implemented by: checks, assertions and triggers)

**Mapping ER diagrams to relational models**【remember to add foreign key/ reference】

Conceptual perspective design: the Entity Relationship (ER) & Logical design (mapping) – the relational model

1. Strong entity – include all simple attributes of itself & Choose one key attribute as primary key (PK)
2. Weak entity – all simple attributes of itself & PK = PK of parent +PK of weak)
3. Binary 1:1 – choose total participation side, add the PK of another side as FK
4. Binary 1:N – choose N-side, add the PK of another side as FK (only 2 don’t need to create a new relations/ table)
5. Binary M:N – include any simple attributes itself & PK of both side as FKs & PK of itself = the combination of FKs

[Notice: 1:1, 1:N can map in the same way as M:N → advantage: reduces ‘null’ that appear as FK values]

1. Multi-valued attribute – PK = multi-value attribute (simple components) and PK of the entity (no non-key attribute)

【V.S. weak entity: weak entity have other non-key attribute】

1. N-ary R’ – PK of all sides as FKs & any simple attributes of n-ary R’ & combination of FKs (with N side) as PK itself
2. Super & subclasses (mapping of EER) – create a relational table for the superclass & each subclass

The PK of each subclass is PK of the superclass (it also is FK)

**Decomposing a relation** (1 → many tables) – replace the relation R by 2 or more relations such that:

Each new relation contains a subset of the attributes of R & Every attribute of R appears in at least one new relation

**Join**: – (natural) join of 2 relations – each ’s tuple links with every ’s tuple having the same values on the common attributes

Lossless-join decompositions: (break a relation, and we get exactly the same R back together with join)

Lossy-join decomposition (loss – loss of information OR addition of spurious information)

**Functional dependencies (FD)** – one attribute determines another attribute

key: minimal set of attributes functionally determines all attributes

Closure of F: the set of all FDs implied by F (closure for a set of attributes X is denoted by )

**Armstrong’s Axioms** ( are sets of attributes): - Reflexivity:

* Transitivity:
* Augmentation:

**Additional rules**: - Union: - Decomposition:

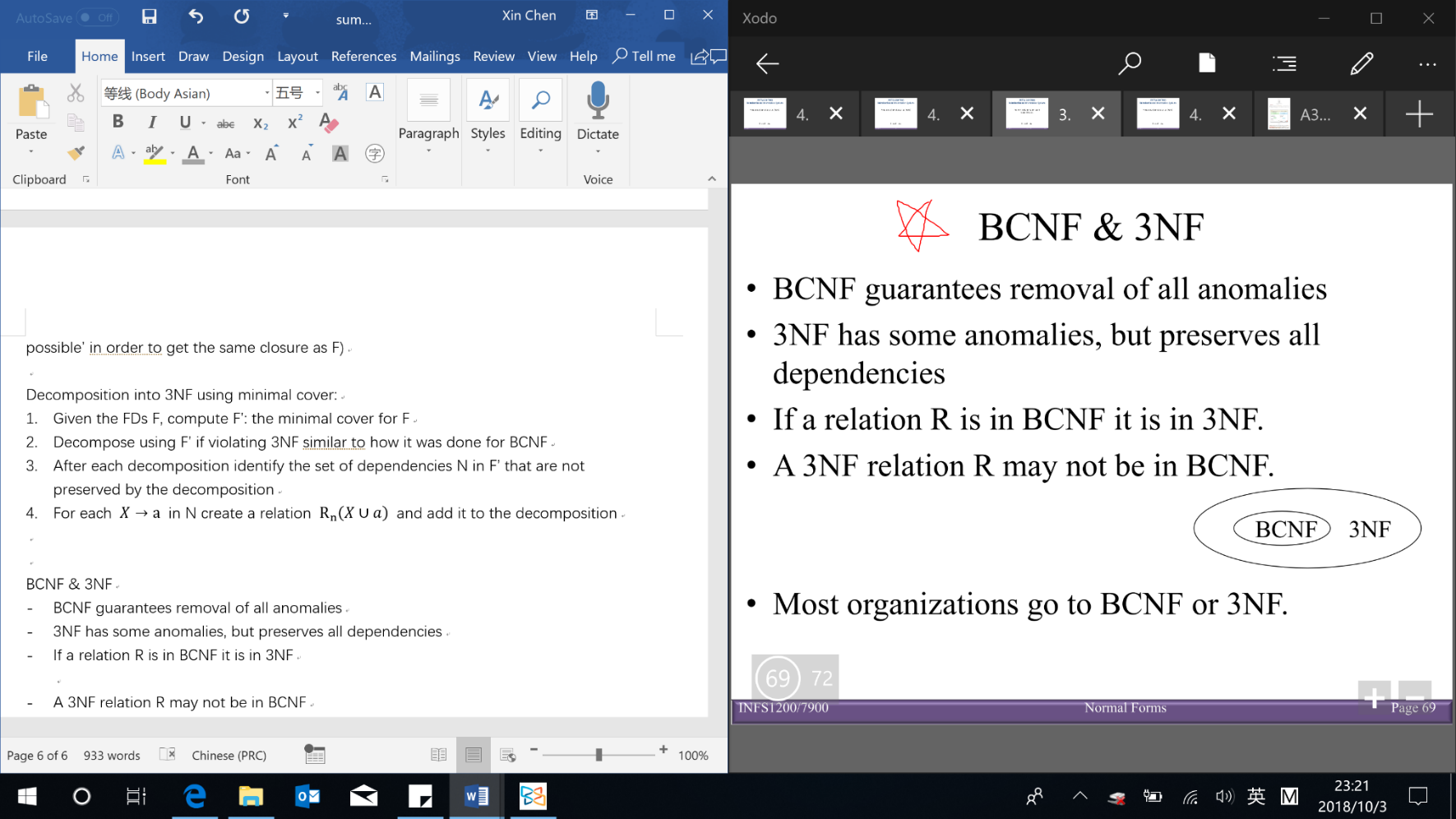
Role of FDs in detecting redundancy (suppose 3 attributes, A, B, C):

* No FDs hold: there is no redundancy here
* Given A → B: several tuples could have the same A value, and if so, they’ll all have the same B value

**Normalization**: process of removing redundancy from data – achieve better designed schemas using FDs and PKs

* **1NF** – only have single value as attribute (BUT have redundancy OR too many NULL value)
* **2NF** – FD: X→Y, where X is minimal key and Y is non-prime attribute, then no proper subset of X determines Y
* **Boyce-Codd (BCNF)** – for every FD: if X → b (non-trivial: b is not the subset of X), then X is a Superkey for R

【LHS is Superkey, RHS is not part of LHS】√: no redundancy&anomalies ×: cannot preserve all dependencies

* **Third normal form (3NF)** – (RHS is not part of LHS) LHS is Superkey OR RHS is prime attribute

√: keep some redundancy/anomalies x: preserve all FDs

Minimal cover (G) of set F: RHS is single attribute & cannot delete any FD or attribute

Denormalization – violate normal form to gain performance improvements: fewer joins & reduces number of FKs

**SQL (structured query language)** – declarative, based on relational algebra【result of SQL query is table/relation】

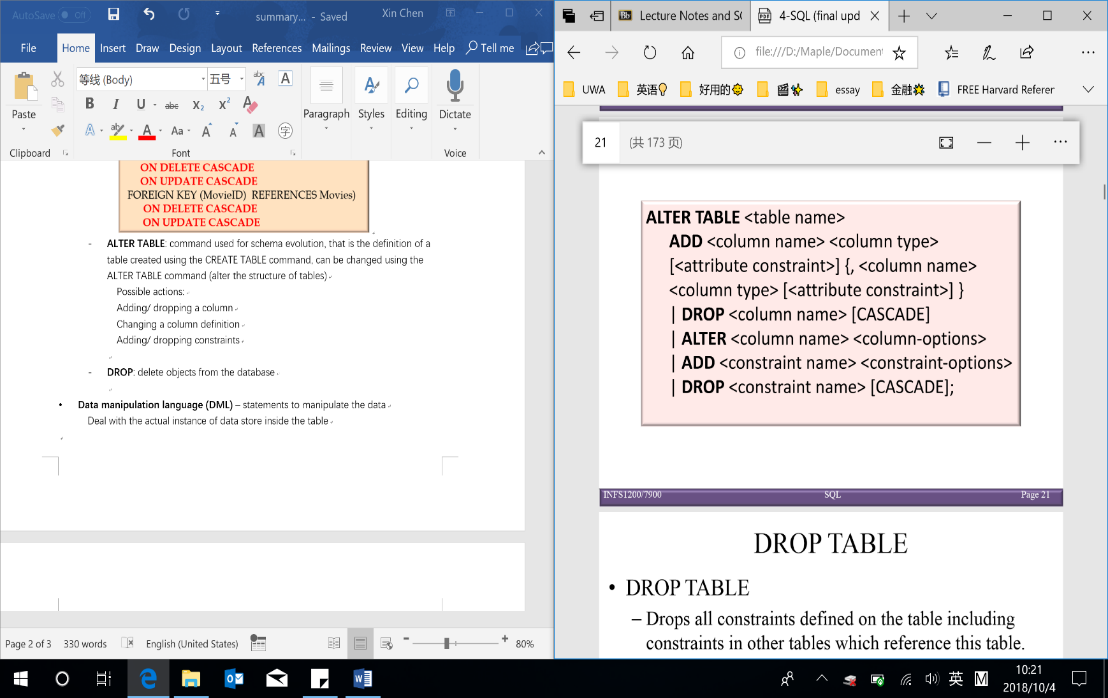
**Data definition language (DDL)** – define database schema/ structure – e.g. CREATE, ALTER and DROP TABLE

**CREATE TABLE**: create a new relation (specifying table name, attributes and constraints)

Domain constraint specified for each attribute – can specified directly or by CREATE DOMAIN

Enforcing referential IC → referential triggered actions when referenced tuple is deleted or updated:

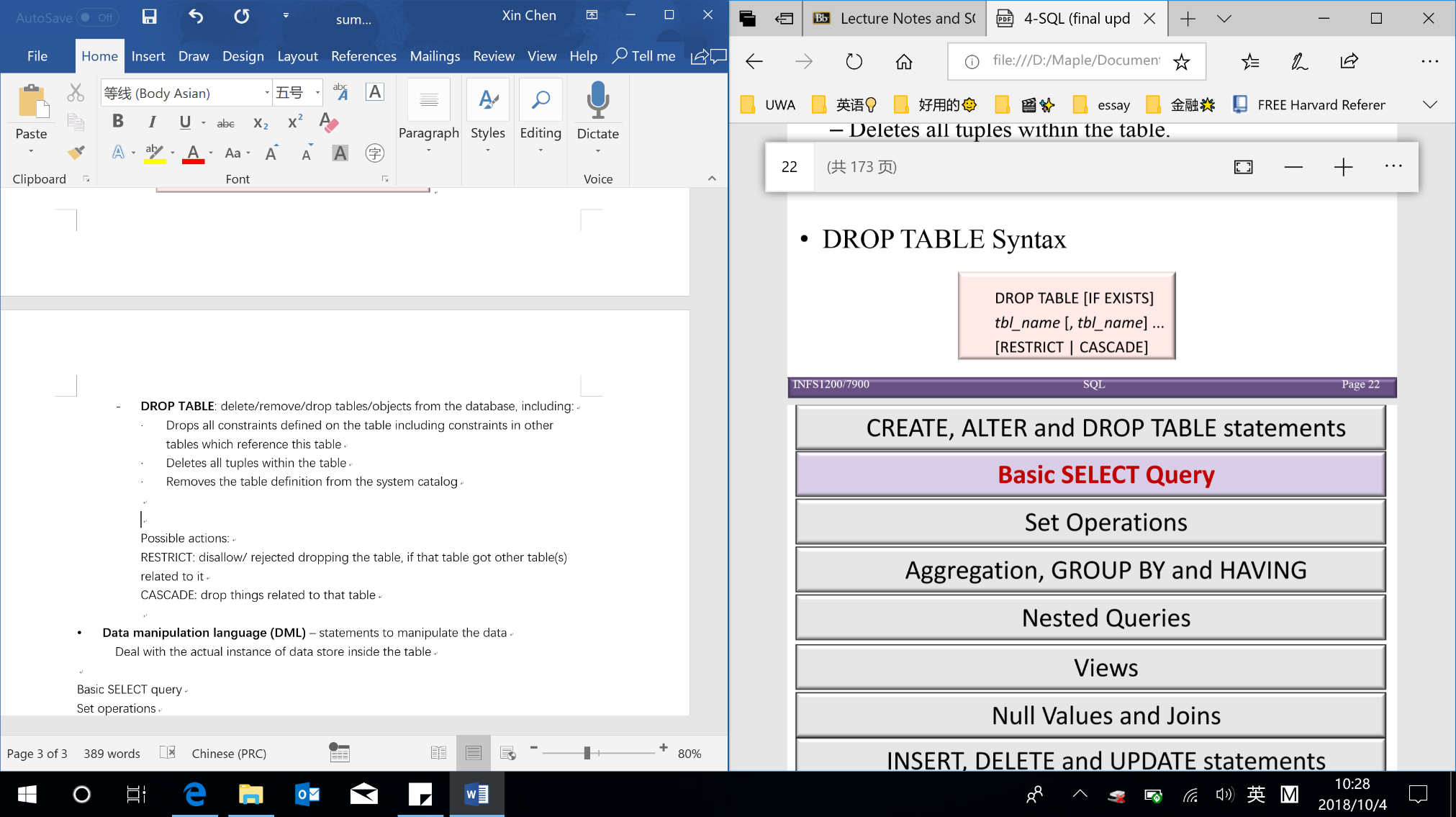
* NO ACTION (default): delete/update is rejected
* SET NULL
* SET DEFAULT
* CASCADE: delete/update all roles that refer to

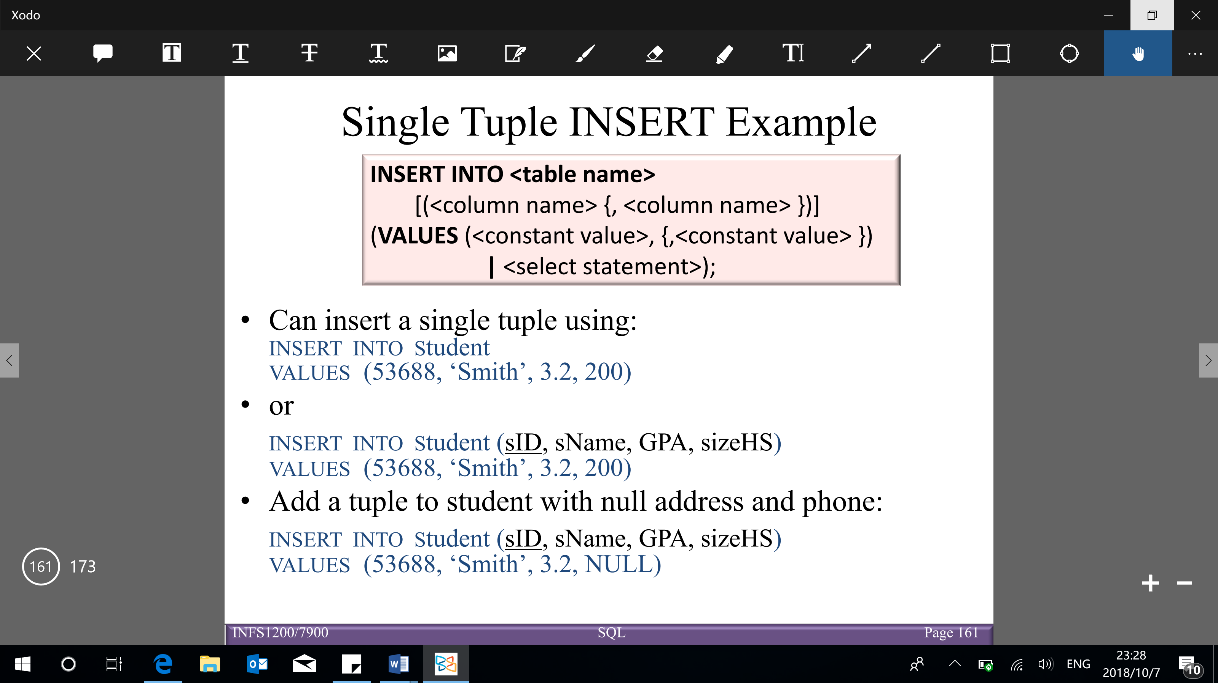
**ALTER TABLE**: alter the structure of tables

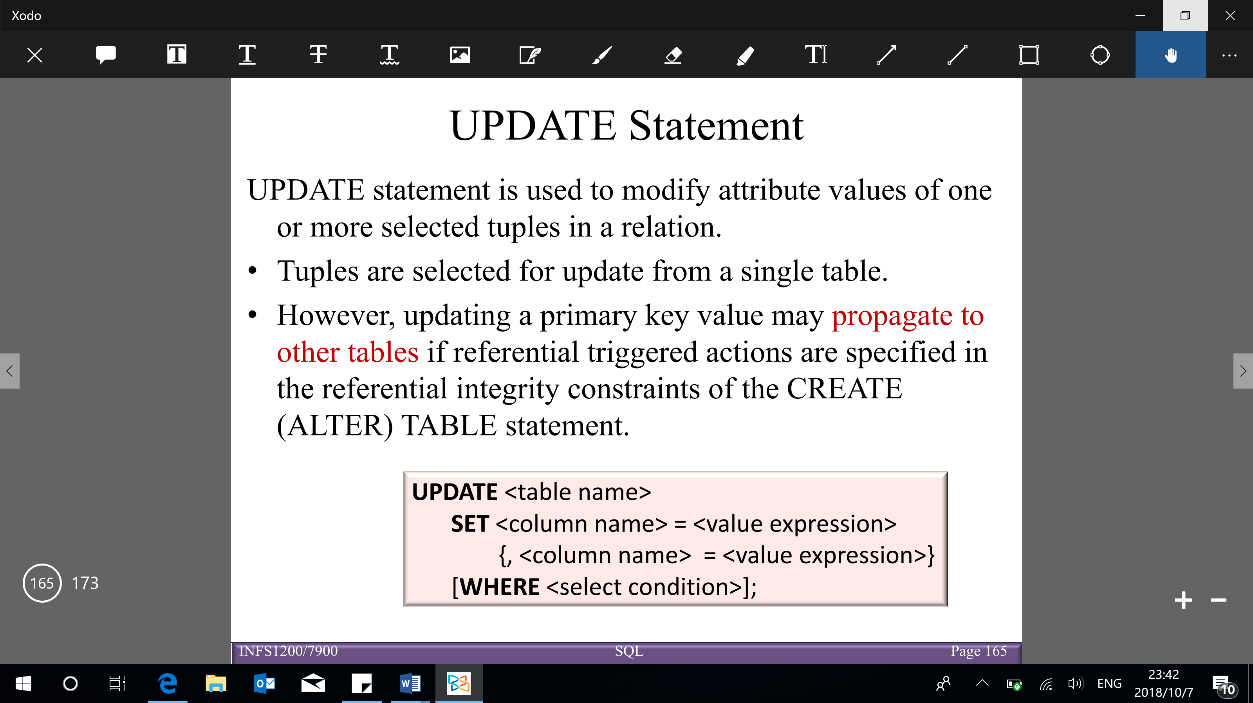
(add/ drop column OR change column definition/domain OR add/ drop constraints)

**DROP TABLE**: delete tables (includes all tuples, ICs, FKs in other table) from DB

**Data manipulation language (DML)** – manipulate/ managing data

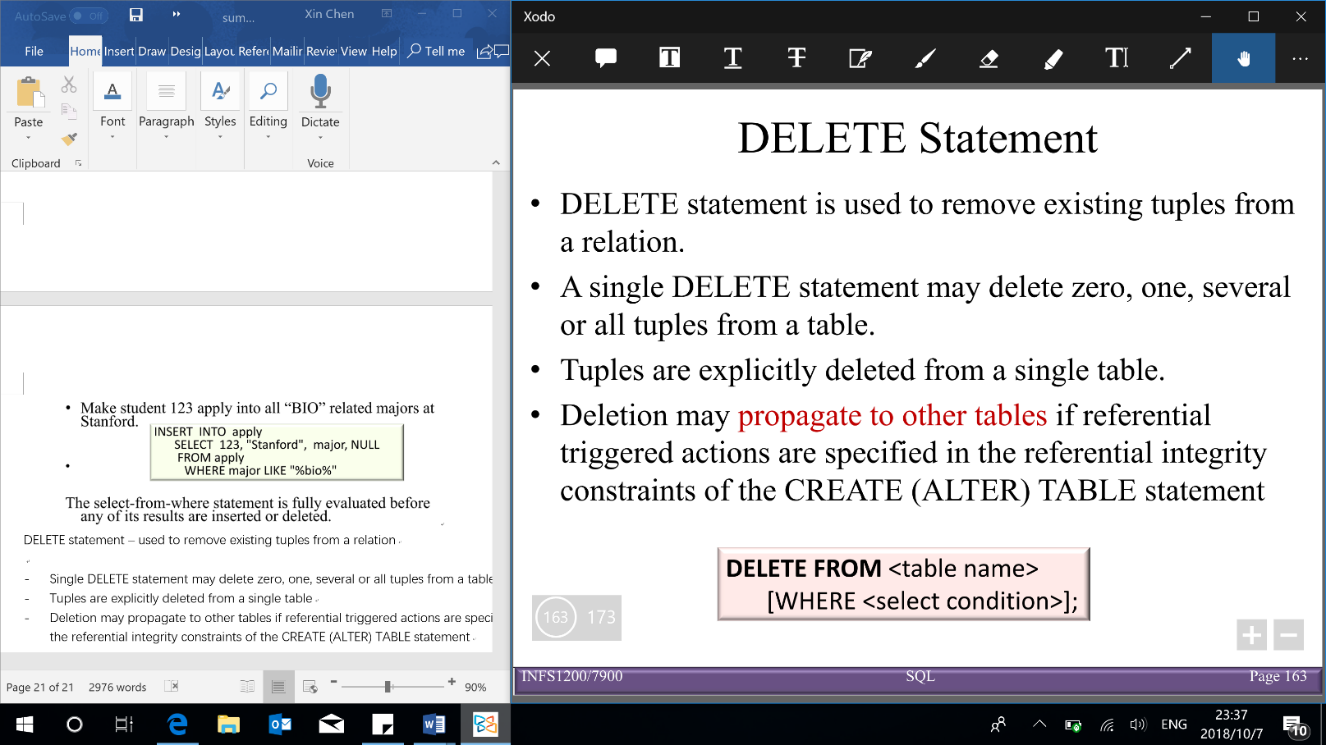
**SELECT**: retrieve data from a database; **INSERT**: insert data into a table; **UPDATE**: updates existing data within a table; **DELETE**: deletes records from a table

**Relational algebra**:

* **Projection (SELECT)**: vertically select attributes wanted to return

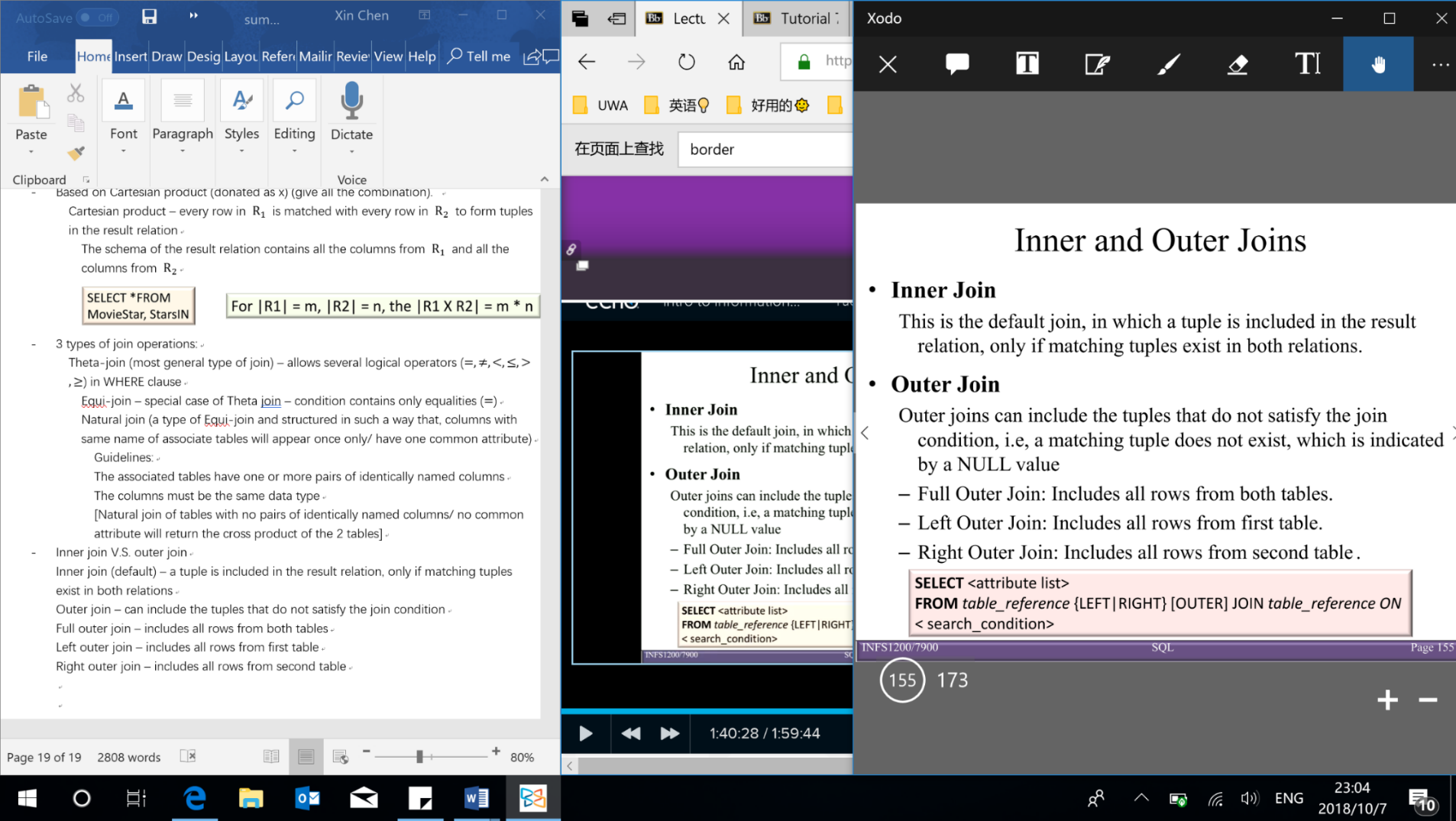
Can evaluate expressions (+, -, \*, /) (e.g. SELECT Year+2)

Rename attributes: *old-name* AS *new-name* (e.g. SELECT Role AS Role1)

* **Selection (WHERE: join condition & search condition)**: horizontal scanner (select tuples)
* **Join (FROM & WHERE join condition)**

– join 2 tables: R1, R2 on their shared attribute

Based on Cartesian product (give all the combination) – rows in , rows in → with rows

Types of join operations (3) in WHERE clause:

Theta-join: with ()

Equi-join: with (special case of Theta join)

Natural join (Equi-join): columns with same name of associate tables will appear once only [if having no pairs of common attribute → return cross product of 2 tables]

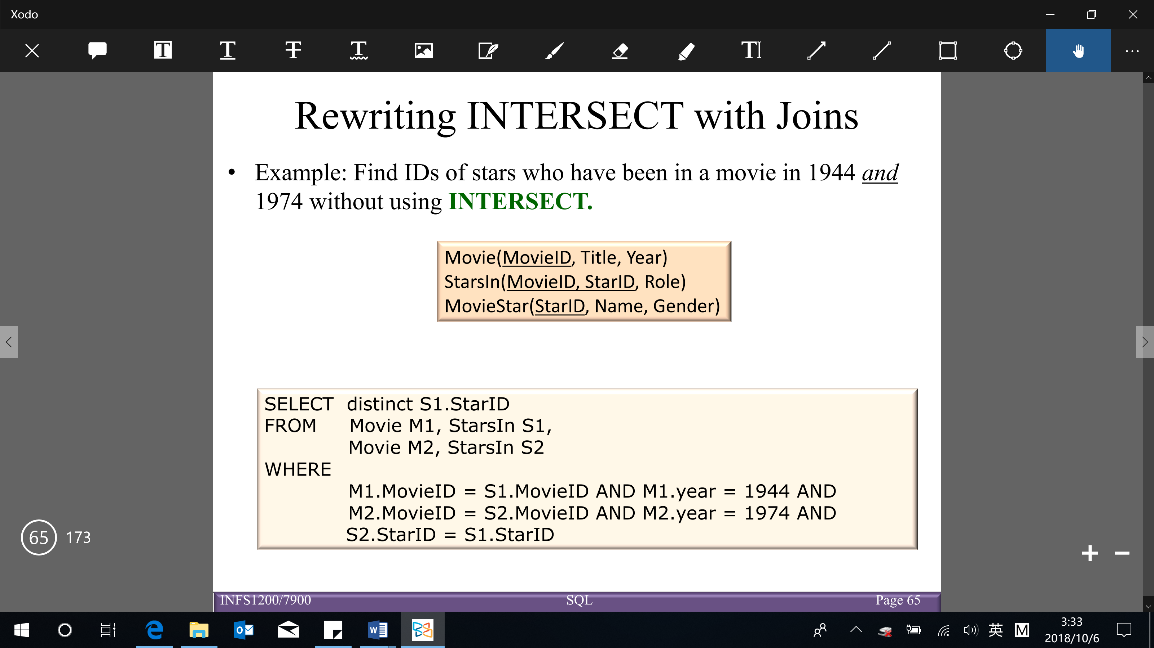
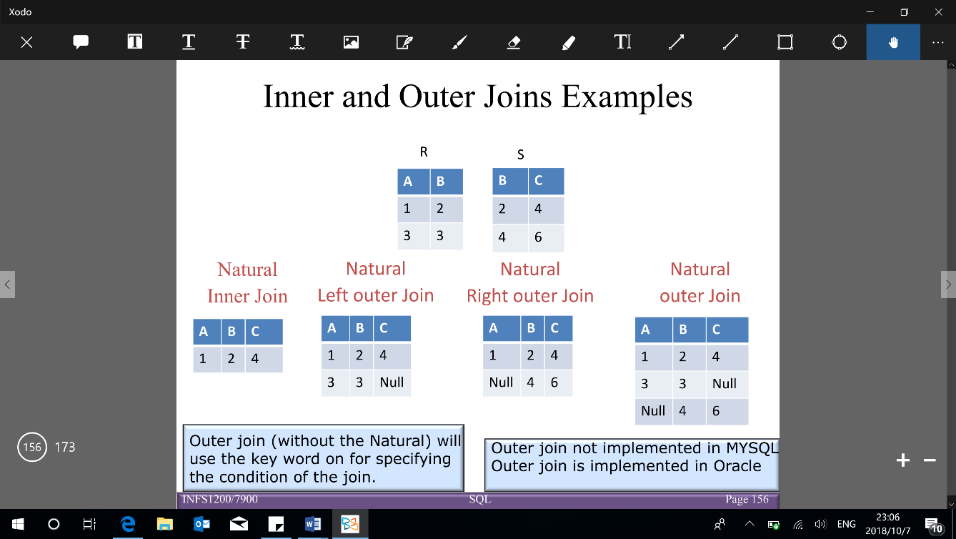
Inner join (default) – includes the tuple only if matching tuples exist in both relations

Outer join – include tuples that do not satisfy the join condition [set NULL value to tuple that does not match]

* **Sorting (ORDER clause)**: order result tuples according to given sort key [ASC (default): small to large/ DESC]

**Set operations** – relation is set of tuples (no duplicates)【must be **union compatible** to do operations: iff 2 relations have same degree (number of columns) & each column has same domains】

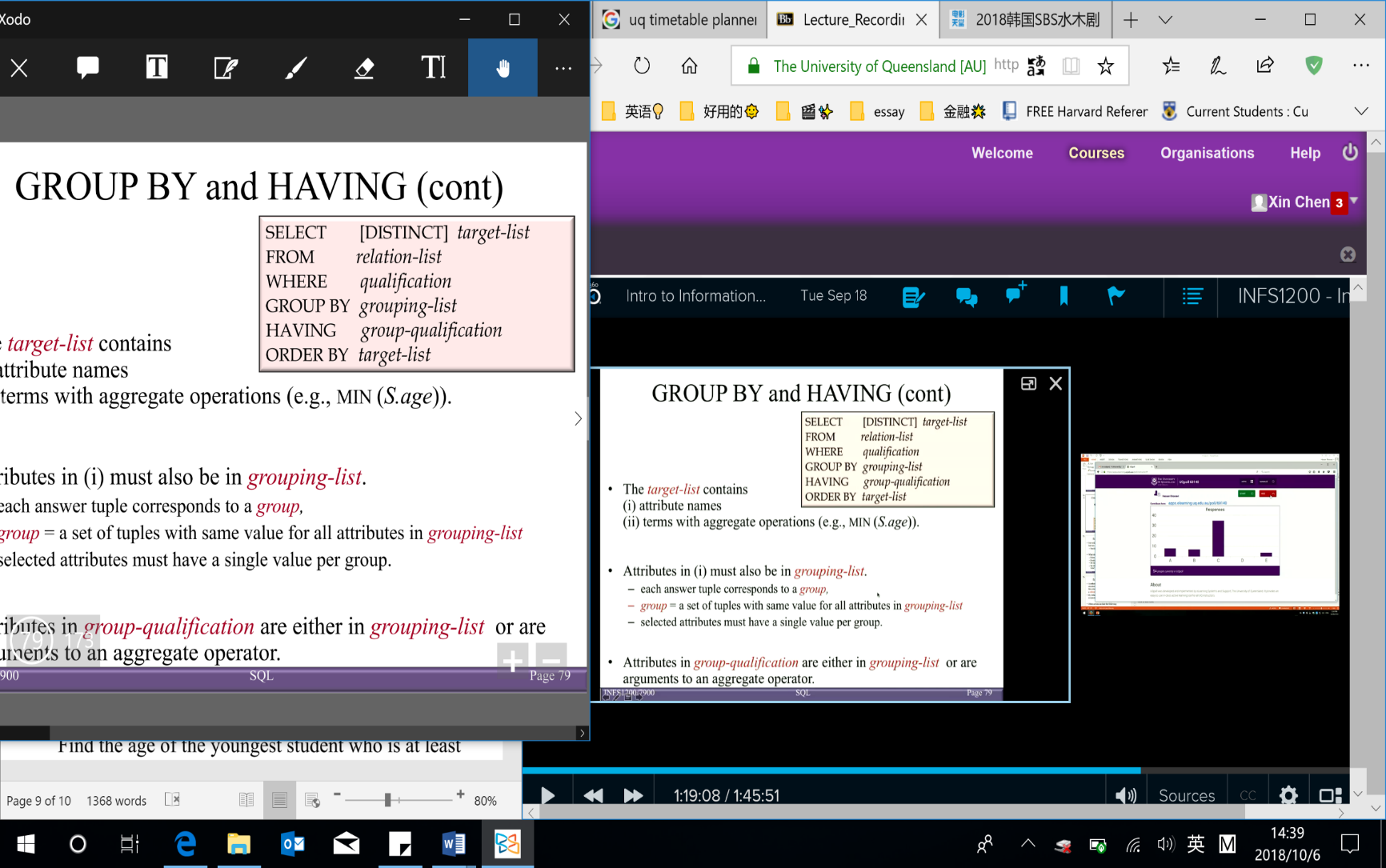
* **UNION ()**: SELECT … UNION [ALL] SELECT … [UNION [ALL] SELECT …] same as OR using in WHERE clause
* **INTERSECTION ()**: SELECT … INTERSECT [ALL] SELECT … [INTERSECT [ALL] SELECT …]

【Note: INTERSECT is part of the SQL standard, BUT is not implemented in MySQL → do it in 2 table】

* **DIFFERENCE/EXCEPT/MINUS ()**: includes all tuples in , but not in 【Note: EXCEPT can use in SQL, BUT cannot use in MySQL → nested queries can work as EXCEPT】

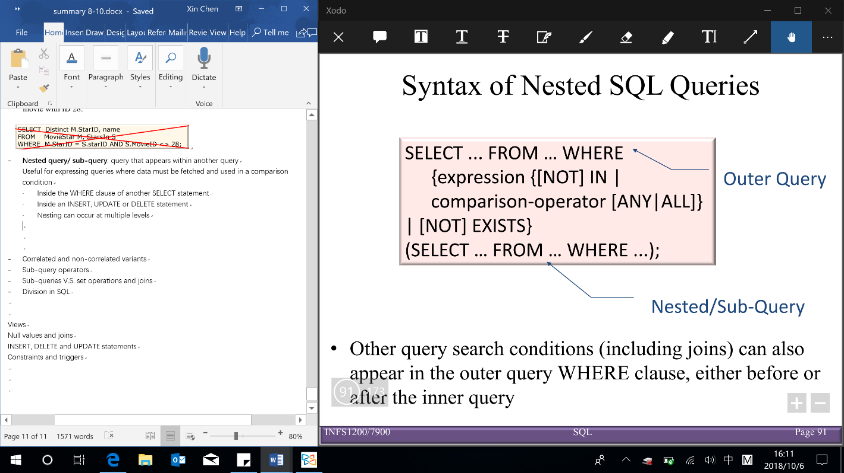
SELECT … EXCEPT [ALL] SELECT … [EXCEPT [ALL] SELECT …]

**Aggregation**: produce summary values on SELECT clause (can’t nested aggregation)

**GROUP BY and HAVING** – divide tuples into groups (GROUP BY) and apply conditions to each group (HAVING)

Conceptual evaluation of a query:

1. Compute cross-product of relation-list (FROM); 2. Keep tuples that satisfy qualification (WHERE); 3. Groups remaining tuples in grouping-list (GROUP BY); 4. Keep groups that satisfy group-qualification (expressions in group-qualification have 1 value per group) (HAVING); 5. Delete fields that are not in target-list (SELECT) → Generate 1 answer tuple per qualifying group; 6. DISTINCT, eliminate duplicate rows; 7. ORDER BY, sort the results

**Nested queries/sub-query**: query within query – useful where data is fetched and used in comparison condition

**·** Inside WHERE of another SELECT statement · Inside an INSERT, UPDATE or DELETE statement · Sub-query cannot include ORDER BY, but DISTINCT may order results of a sub-query

* **Correlated and non-correlated variants (nested queries)**

Non-correlated – inner, outer query run independently (don’t need to do join) → computed just once (evaluated from the ‘inside out’) – e.g. IN, NOT IN & Correlated – inner depends on outer query (need to do join in inner query WHERE) → compute many times – e.g. EXISTS/ NOT EXISTS

* **Sub-query operators** (expression and attribute list in sub-query SELECT clause must have same domain)
  + [NOT] IN (sub-query)
  + Comparison-operator () [ANY|ALL] (sub-q) [can only use when sub-query return 1 value]

Equivalence: ‘IN’ and ‘= ANY’ & ‘NOT IN’ and ‘<> ALL’ & Non-equivalence: ‘NOT IN’ and ‘<> ANY‘

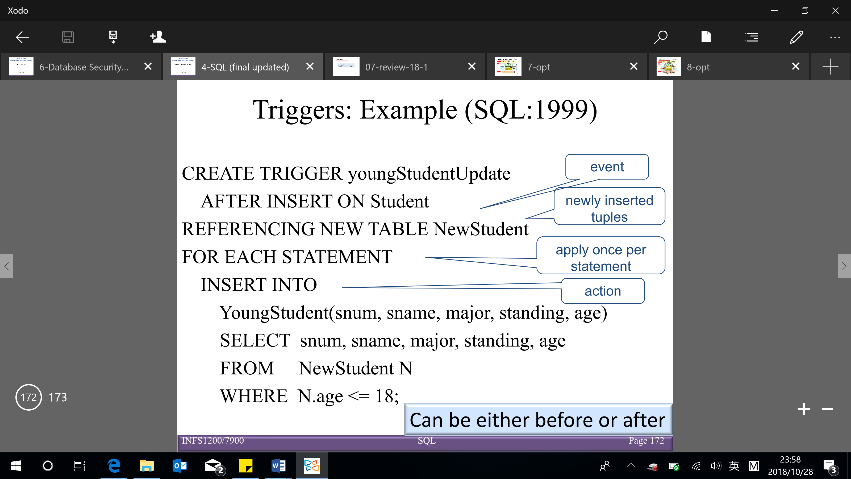
Sub-queries V.S. set operations V.S. joins(can be used to write the same query)

· Joins – display results from multiple tables · Sub-queries – compare aggregates to other values

**Division** – for queries include ‘for all’/ ‘for every’[BUT no direct way to write division in SQL →double negation]

**Views** – types: virtual tables (not physically exist) & materialized (physically exist & need to update)

**NULL value** – indicates value is unknown [IS NULL (IS NOT NULL): used to check whether value is known/unknown]

* Operations on NULL value [NULL requires 3-valued (true, unknown, false) logic using the truth value unknown]
  + OR: (unknown OR true) = true; (unknown OR false) = unknown; (unknown OR unknown) = unknown
  + AND:(true AND unknown) = unknown; (false AND unknown) = false; (unknown AND unknown) = unknown
  + ****NOT:(NOT unknown) = unknown
  + Comparisons between 2 null values, or between a null and any other value → return unknown [All aggregate operations except COUNT(\*) ignore tuples with null values on the aggregated attributes]

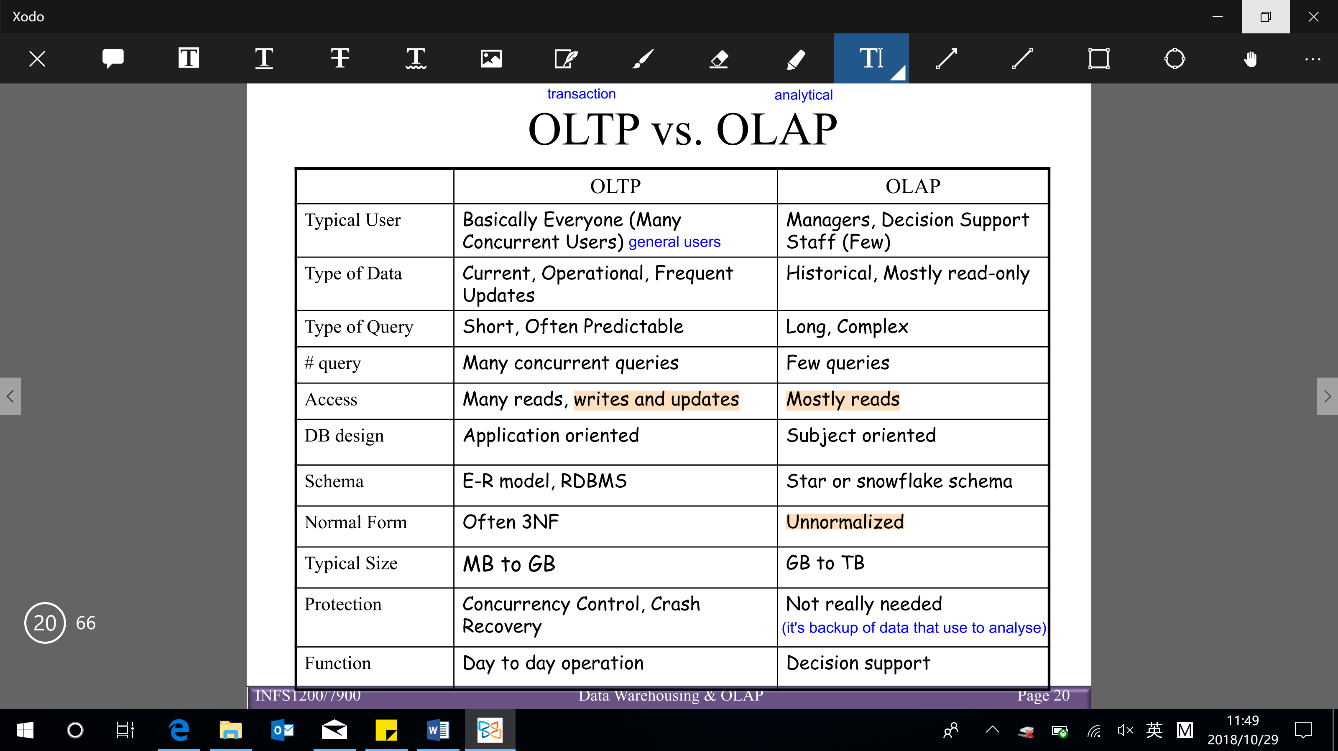
Semantic constraints can be specified using CHECK and ASSERTION statements

* **CHECK**– write at the end of CREATE TABLE (check when tuples are inserted or modified) CHECK (condition)
* **ASSERTION** (like CHECK) – check multiple tables CREATE ASSERTION <name> CHECK (NOT EXISTS (SELECT …))
* **TRIGGER** – procedure that start automatically if specified changes occur in DBMS (may cause cascading effects)

**Data warehousing** – process of constructing and using data warehouses

Different types of needs: - Operational DBMS (for running business) – OLTP - Data warehouse – OLAP

OLAP (on-line analytical processing): technology that enables analysts, managers, and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information

**Star schema** – one table for fact table (BCNF) + one table per dimension (not normalized/ redundant – fewer joins)

**Snowflake schema** – one fact table + dimensions (one table per level – normalized)

Roll-up/ drill-down – like zoom in/out; Slicing – look into 1 value in one of the dimensions but not zoom in/out (use ‘=’)

Dicing – look into one of the values in multiple dimensions

Pivoting – rotate cube to provide an alternative presentation (changing GROUP BY items)

WITH ROLLUP/CUBE (end of GROUP BY) – combine fact data and dimensions [Roll-up, drill-down, slicing, dicing, and pivoting operations are expensive → CUBE pre-calculated]

Number of tuples: (Just GROUP BY all dimension: )

ROLLUP: [GROUP BY A, B, C WITH ROLLUP (order matters), can do by UNION]

CUBE: [ for computing all cells for a k-dimensional cube, can do by pivoting ROLLUP]

Database security

**Privacy**: ability of individuals to control the terms under which their sensitive data is acquired and used

**Security**: a required building block for privacy (reasons: legal/policy, ethical, technical), which includes:

- Preventing storage of sensitive data - Ensuring appropriate/ authorized (access of people) use of sensitive data

Database control measures:

1. Access control (3 mechanisms) – create user accounts and passwords (& give different privilege to accounts)

Discretionary access control: grant privileges to users (2 level): 1. Account level (high level) – CREATE SCHEMA/ TABLE; 2. Relation level – access matrix model: owner can grant (select, write and references) through views

Grant privilege: [GRANT SELECT ON V TO B;] (SELECT – read access)

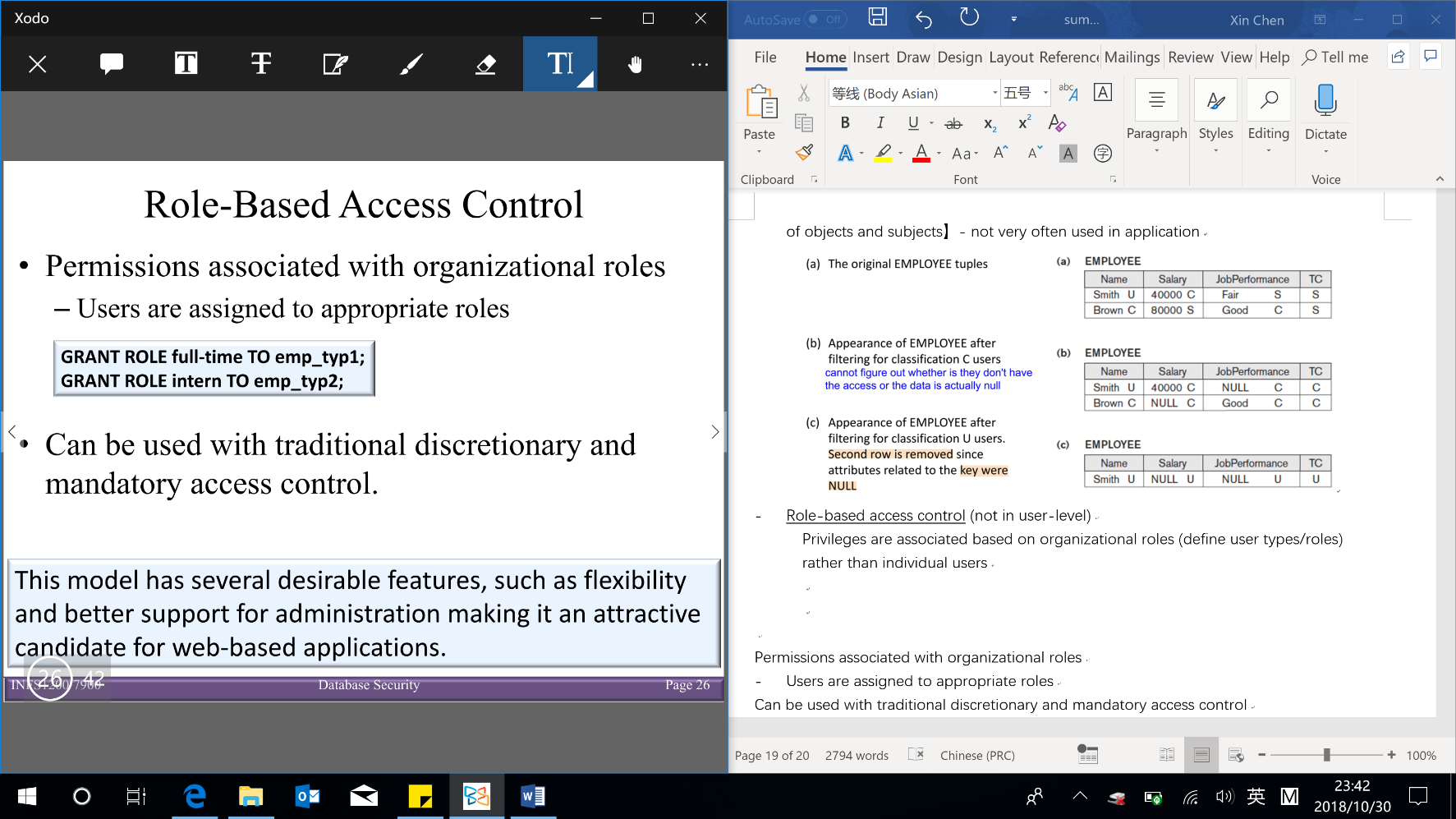
Revoking of privileges: REVOKE command used to cancel a privilege [REVOKE SELECT ON V FROM B;]

Spread of privileges using the GRANT OPTION: [GRANT SELECT ON V TO B WITH GRANT OPTION;]

Mandatory access control: classify data and users into security classes (Top Secret, Secret, Confidential, Unclassified)

* + Simple security property (intuitive) – allowed to read
  + Star property – allowed to write, if (can only write things to its class) → Prevent information from flowing from higher to lower classification 【the row is removed if the key is NULL】

Role-based access control (not in user-level) – privileges are based on organizational roles (define user types/roles) → users are assigned to appropriate roles (can use with discretionary and mandatory access control)

√: flexibility and better support for web-based applications

1. Inference control – information about individuals cannot accessed (only statistical queries are allowed) · Provide minimum threshold on number of tuples · Prohibit sequences of queries that

refer to the same population of tuples · Introduce slight noise or inaccuracy in database but maintain the correct statistical data · Partition the database – store records in groups of minimum size (new tables)

1. Flow control – prevents information from flowing to unauthorized users
2. Data encryption – used to protect sensitive transmitted data (protect data while it’s in transaction)

Basic DBMS security functions: 1. User accounts: user log in with username and password; 2. Login session: sequence of database operations by user recorded in system log (log store the info on the users); 3. Database audit: reviewing log to examine all accesses and operations applied (see who make a change)

SQL injection methods:

1. SQL manipulation: changes an SQL command in the application (e.g. OR ‘x’=‘x’) to attack occurs during login
2. Code injection: add additional SQL statements or commands that are then processed
3. Function call injection: database or operating system function call inserted into vulnerable SQL statement to manipulate data or make a privileged system call