-Entity – has multiple instances (table)



-Attribute – characteristic of an entity

-Always prefix table name with same word

-Never include foreign keys, derived data

-Put attributes with variable occurrences in separate table -remember to decompose composite attributes

-Exceptions to numeric primary keys: SKU, ISBN, VIN -Relationships = 1:1, 1:Many, Many:Many

-Always add linking tables for Many:Many (these are composite relationships – relationships with attributes)

-Relationships degrees = unary, binary, ternary,



Generalization/specialization

-Homonym – different attributes, different tables, same name (avoid)

-Synonym – same attribute has different names in different tables

(FK should have same name as PK)

-Data Model Levels – conceptual (ER model), logical

(relational table design), physical (DDL’s)

-Database integrity – entity integrity (every record has unique, not null PK), referential integrity (every FK exists as a PK)

-PK always goes on the “1” side of 1:Many relationship

-For generalization/specialization, PK of sub-table is also an FK

-Insertion anomalies – inserting a new row makes data inconsistent, can’t insert a new row because its primary key

can’t be defined yet

-Update anomalies – you fail to update all instances of a data value

-Deletion anomalies – you delete a row and inadvertently delete other data you don’t want to delete

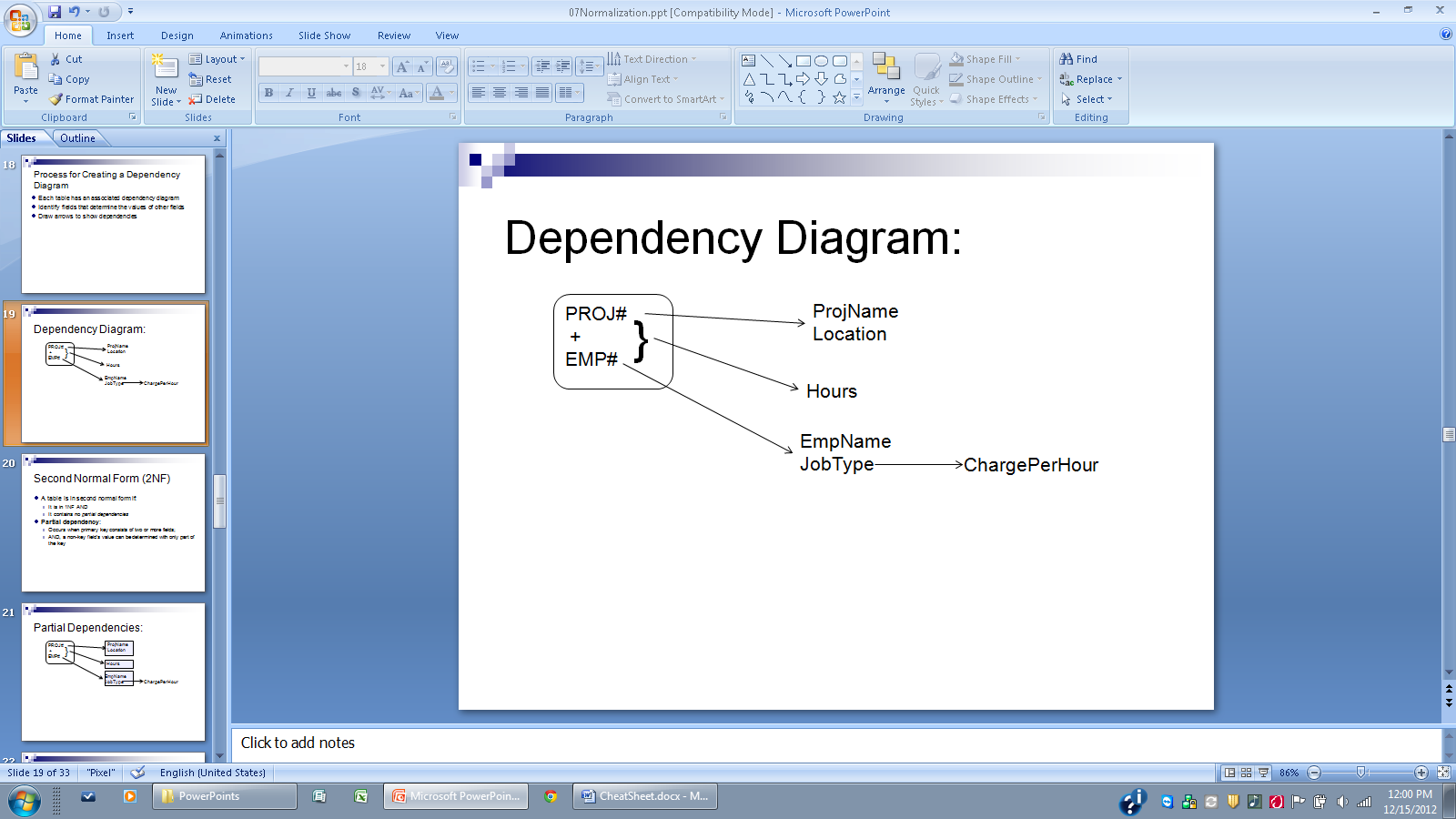
-1NF – no repeating groups within table cells (each cell contains only 1 atomic data value)

-2NF – 1NF and contains no partial dependencies (when a PK consists of >= 2 fields & a non-key field’s value

can be determined with only part of the key)

-3NF – 2NF and no transitive dependencies (non-key field determines the value of other non-key field)

-Candidate keys – minimal set of attributes that uniquely identifies a row of data (can be multiples), cannot

 depend on anything else

-stored program – stored in database/automates data processing – written in language specific to database

-why: return results as tabular set, formatting, loops/if’s, faster than external languages, uses DBMS security

-why not: increases DBMS server’s load, lack of language flexibility, difficult to debug

-PL/SQL not case sensitive -Comments: --single line /\* block of comments \*/

-SET SERVEROUTPUT ON SIZE 4000;

-IF THEN, ELSE, END IF; -IF THEN, ELSEIF THEN, ELSE, END IF;

-WHILE condition -LOOP

LOOP statements;

statements; EXIT WHEN condition;

END LOOP; END LOOP;

-FOR counter IN startValue .. endValue -implicit cursor: SELECT INTO

LOOP

statements;

END LOOP;

-Exceptions: predefined, undefined, user-defined

-Undefined exceptions handled w. oracle error code

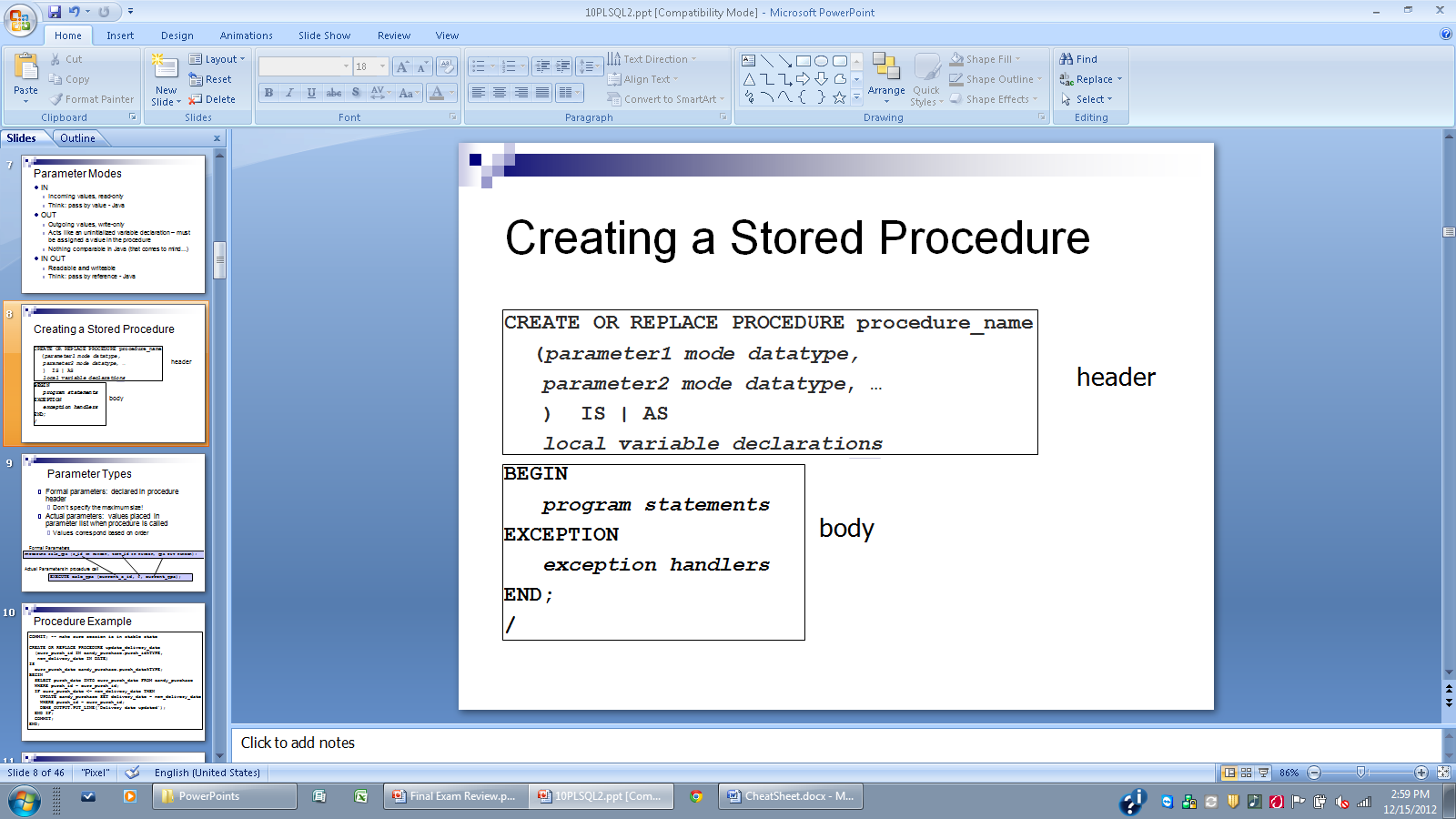
-WHEN OTHERS THEN

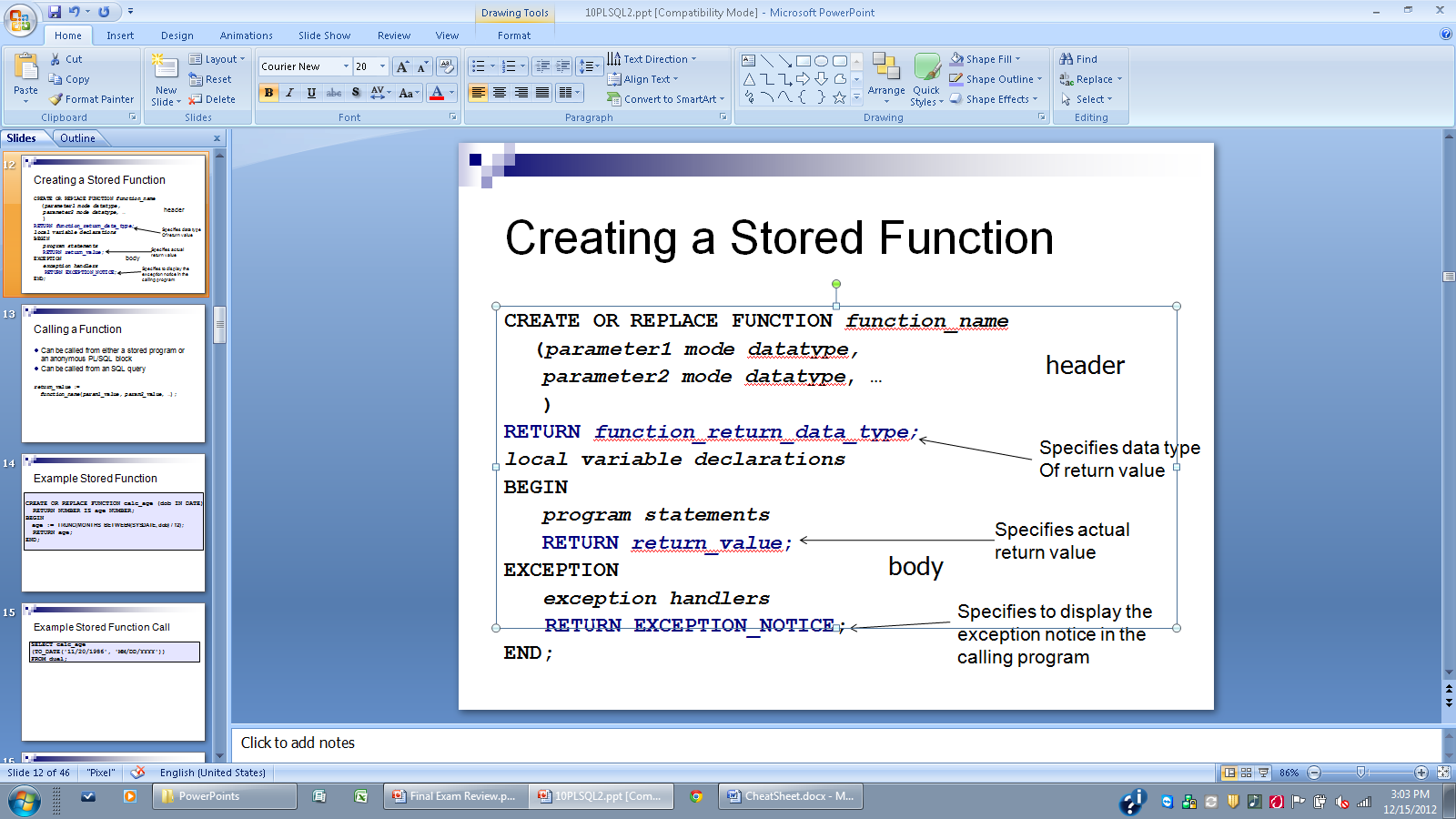
….

-Exceptions only work in their BEGIN/END block

-for named exceptions, replace OTHERS with the name

-Parameter Modes: IN, OUT, IN OUT(pass by reference)





-procedures: receive/pass multiple parameter values

-function: return single value

-Package: server-side code library, organizes stored programs, specification (visible) and body (hidden)

-CREATE OF REPLACE PACKAGE (or PACKAGE BODY)

-stored program: object in schema, stored in compiled form

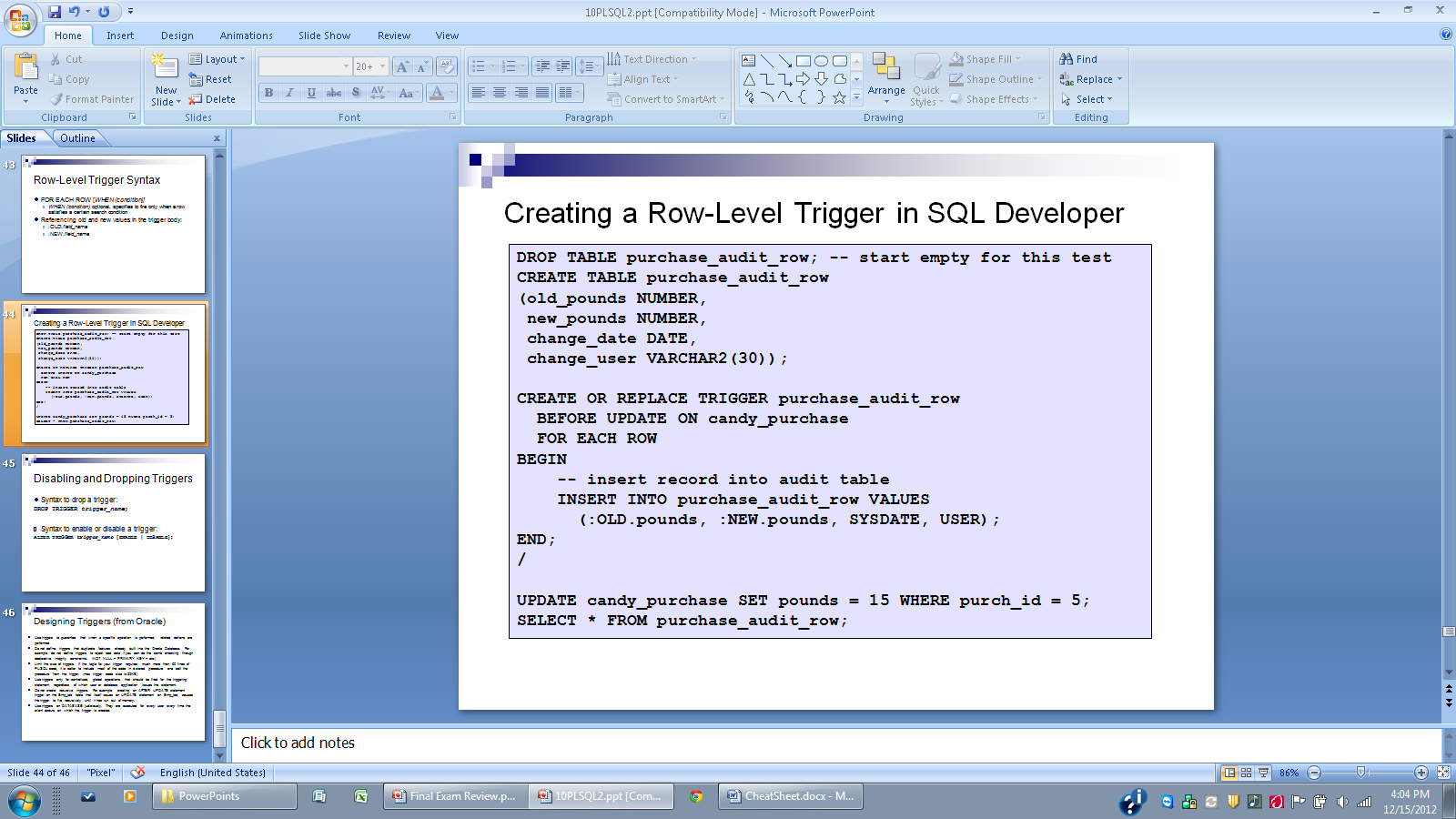
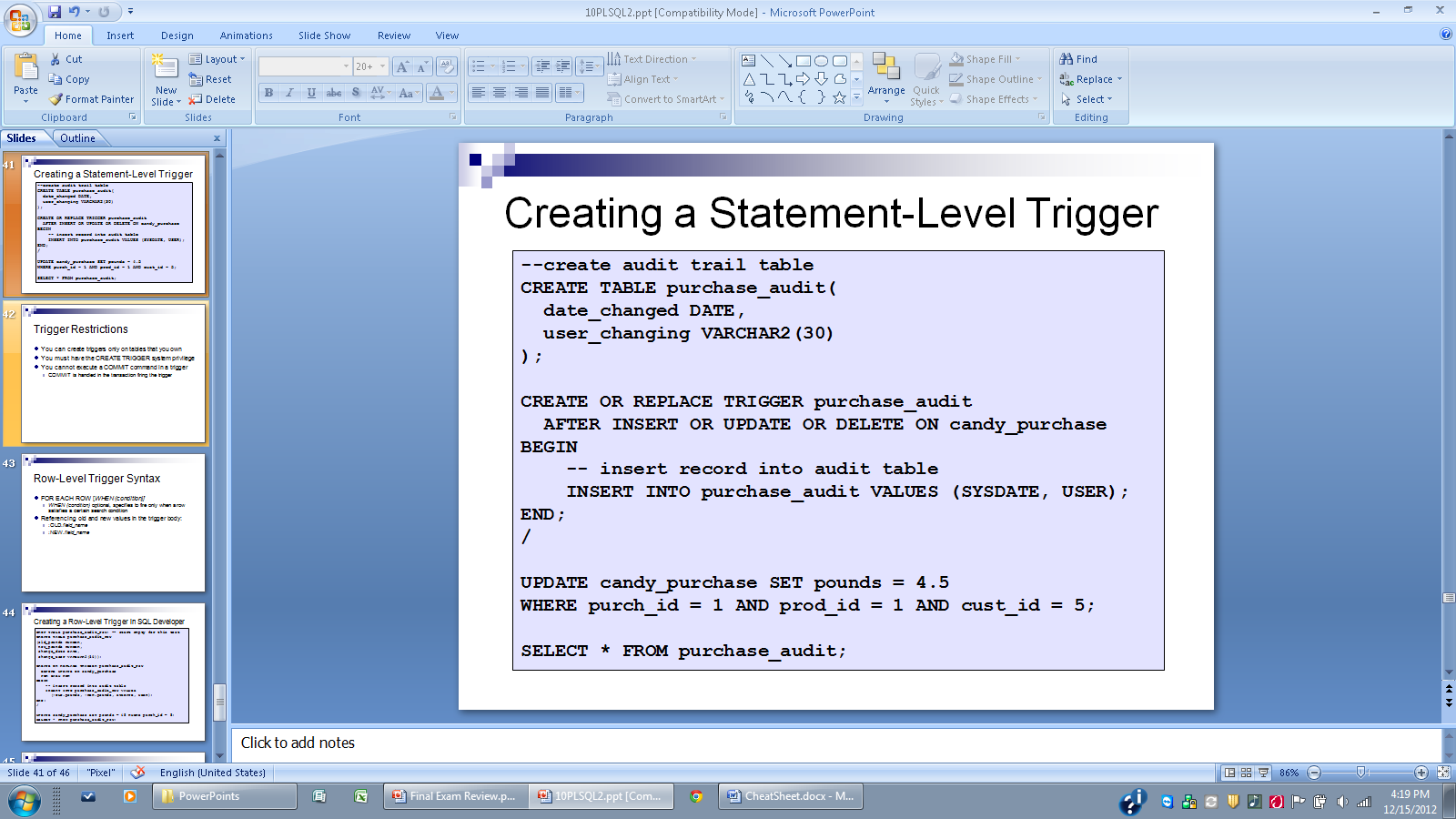
-combine blocks in a script by adding a “/” between them

-Triggers: program units attached to specific table, execute in response to INSERT, UPDATE, and DELETE,

force related operations to always happen/create an audit trail, cannot accept parameters, cannot commit inside

-for triggers, specify INSERT/UPDATE/DELETE, BEFORE/AFTER, STATEMENT/ROW

-DROP TRIGGER trigger\_name; -ALTER TRIGGER trigger\_name [ENABLE | DISABLE];



-row level trigger syntax: FOR EACH ROW [WHEN (condition)]

Use “:OLD.field\_name” or “:NEW.field\_name” to refer to old/new values in trigger body

-transaction – logical unit of work (>= 1 INSERT/UPDATE/DELETE command)

-transaction goals: **atomic** (cannot be subdivided/all or none succeed), **consistent** (all application constraints true before are also true after), **isolated** (changes not visible externally until transaction ends), **durable** (changes are permanent)

-Implicit transaction control – Oracle default, starts with connection, ends with COMMIT (or when you close session)

-Auto commit – every action query (can’t roll them back), MySQL default (not Oracle), single command transactions

-Explicit transactions **(only in MySQL)**– START TRANSACTION;

COMMIT;

-Explicit transactions can be created in Oracle by using PL/SQL blocks

-SAVEPOINT savepoint\_name; -ROLLBACK TO savepoint\_name;

-Lost updates – many transactions read a record, then update it – all updates except last one are lost

-Dirty reads – reading a record with uncommitted data, which

Is then rolled back

-Nonrepeatable read – multiple reads return different data

(updates/deletes)

-Phantom read – multiple reads return new/missing records

-lock types – read / write(UPDATE/DELETE)

-OF fieldname only supported in Oracle -NOWAIT – MySQL default

-MySQL: SET autocommit=0; -Oracle/MySQL both automatically fix deadlocks

-Lock lvls – record lvl (default), table lvl (explicit) -Lock modes – shared (prevents other exclusive locks), exclusive

(prevents other shared and exclusive locks)

-Isolation lvl – how DBMS enforces row lvl locks

-Read uncommitted – transactions don’t acquire locks for anything, use only when data is never modified, nothing gets

blocked, and nothing is critical, allows all issues

-Read committed – transactions acquire exclusive write locks for INSERT/UPDATE/DELETE/SELECT FOR UPDATE, stops

lost updates/dirty reads, allows non-repeatable reads/phantom reads, Oracle default

-Repeatable read – MySQL default, transactions acquire shared read locks for all SELECTS, avoids all issues

-Serializable – in MySQL transaction blocks all competing INSERTS/UPDATES/DELETES, in Oracle transaction doesn’t

see other session’s committed INSERTS/UPDATES/DELETES (like repeatable read)

-Database performance measures – response time, throughput

-database -> tablespaces -> datafiles

-new table records stored in 1st available datafile segment, so data is mixed up and slow

-lots of transactions slow database - queries are serialized, locking and blocking

-index – separate table (binary tree) that contains >= 1 field index values & the ROWID that identifies the associated

record’s local, contains data object #, data block in datafile, position of row in datafile, datafile in which the row

resides

-PK’s and fields with UNIQUE constraints are automatically indexed

-Indexes: simple(single-field), composite(multiple-field), unique(index field must be unique), clustered(records are

forced to be stored in same order that they appear in the index)

-Create index after table is populated with data

-DROP INDEX index\_name;

-Benchmarking queries: ALTER SYSTEM FLUSH BUFFER\_CACHE; ALTER SYSTEM FLUSH SHARED\_POOL;

-indexes only help when query retrieves < 15% of table rows

-Denormalization – create summary table with common queries that join many tables, use triggers to update

summary table, **or use materialized views**

-Materialized views – real table, faster, lots of system overhead, INSERTS/UPDATES/DELETES take longer

-to make read only, omit “FOR UPDATE”

-Distributed databases – networked servers running independent DBMS instances that work together, fragmentation

Is transparent to users

-Full Replication – every node runs same DBMS, has same data

- publisher contains master copy, subscribers receive updated copies from publisher/deliver to users

- Snapshot – publisher distributes snapshot of entire database to all subscribers

- Transactional – changes made only to publisher, publisher distributes changes to subscribers in time

- Merged – changes made to publisher and subscribers and are merged periodically

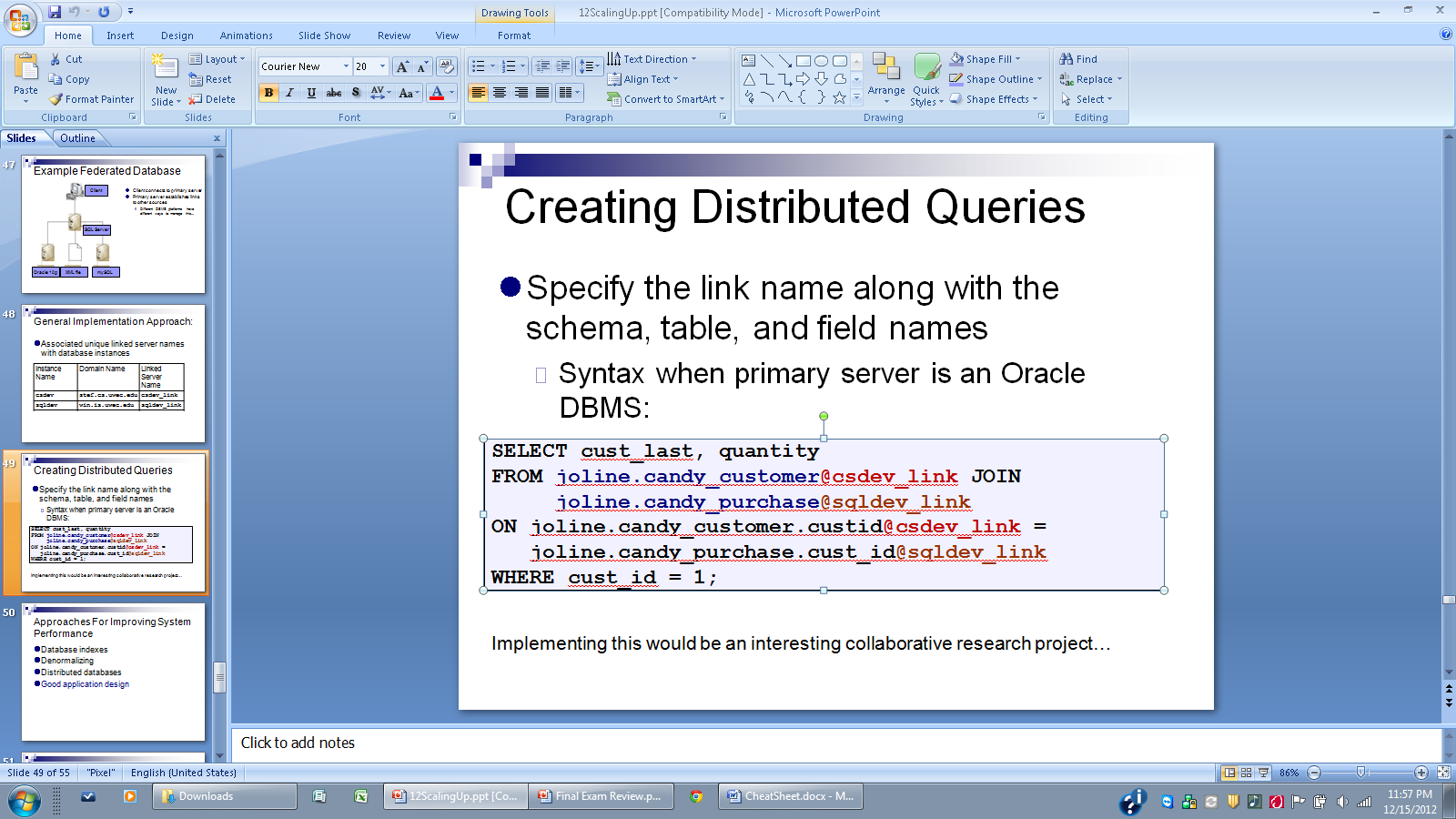
-Homogeneous – nodes run same DBMS, contain different data at different physical locations

- Horizontal fragmentation – all fields included at each local, appropriate records stored at each local

- Vertical fragmentation – appropriate fields stored at each local

-Federated/Heterogeneous – nodes run different DBMS’s, can contain distinct databases with different structures

- agree to share data, syntax is below



-Bottleneck is usually not in database -Create stored procedures as much as possible

-Asynchronous Query – immediately displays some records, while a thread retrieves more in the background

-Synchronous Query – all records must be retrieved before any are displayed

-Use asynchronous query whenever many results are returned, implement it at application level

-When throughput/response time is a problem – test, create indexes, denormalize, modify application, create

distributed database