INFO20003 Tutorial – Week 10 Solutions

(Tutorial: Database Administration and Transactions)

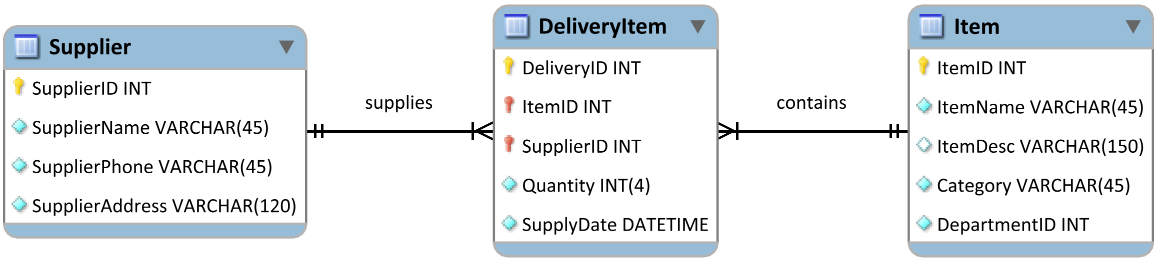
Objectives:

This tutorial will cover:

I. Apply capacity planning concepts – 10 minutes
II. Review of backup & recovery concepts – 15 minutes
III. Apply backup & recovery concepts to case studies – 10 minutes
IV. Review transactions concepts – 10 minutes
V. Apply transactions concepts – 15 minutes

Exercises:

1. Capacity planning



Consider the case of a department store. An analyst has determined that there are 50 distinct suppliers that provide 2000 distinct items to the store. They have determined that the average delivery is of 40 distinct items and that each supplier delivers approximately once a week (the analyst has estimated this to be 50 deliveries a year). For each delivery by a supplier, there are on average 40 rows added to the DeliveryItem table. While the Item and Supplier tables stay constant in size, the DeliveryItem table grows by 100,000 rows every year.

This assumes that suppliers and items stay constant; however, if the business is successful, the suppliers and frequency of deliveries and number of distinct items delivered can be expected to grow. If we know the length of each row, we can estimate how big each table will be year by year.

Using the MySQL information about data type storage and information from the data dictionary, the analyst has determined the following average row lengths of each table:

Table	Number of rows	Average row length
Supplier	50 rows	144 bytes
Item	2000 rows	170 bytes
DeliveryItem	0 rows	19 bytes

Table 1: Row volume and row length for the Supplier delivers Item entities.

Assume that the number of suppliers and number of items do not change from year to year, and that the delivery schedule remains the same. Calculate the size of the three tables:

a. When database use begins (year 0)

Supplier = 50 × 144 bytes = 7200 bytes (approx. 7 KB – *Note: 1 KB = 1024 bytes*)
Item = 2000 × 170 bytes = 340,000 bytes (approx. 332 KB)
DeliveryItem = 0 × 19 bytes = 0 bytes

b. After one year of database use

Supplier and Item remain unchanged.
DeliveryItem = 100,000 × 19 bytes = 1,900,000 bytes (approx. 1.8 MB – *Note: 1 MB = 1024 KB = 1,048,576 bytes*)

c. After five years of database use

Supplier and Item remain unchanged.
DeliveryItem = 500,000 × 19 bytes = 9,500,000 bytes (approx. 9.1 MB)

Tables such as the DeliveryItem table here or the Order-Item table in the Order/Order-Item/Item table schema are known as *event tables*, because they record events (sales, deliveries, orders, academic results). It is the event tables that need to be given special attention when applying capacity planning concepts, because the other tables are not expected to grow rapidly.

Key Concepts:

- Why do we need a backup of our database?

  Backups of databases are required so that in the event of failure of the database system there can be a full recovery of the database so that there is no loss of data. If we cannot make a full database recovery, we may end up with data integrity errors and data mismatch.

- Backup planning
  - Backup type – logical or physical

    Logical – Logical backups save rows as they are displayed to users and the associated metadata information (column name, data types, indexes columns, type of indexes). Not only do we keep a record of the data in a logical backup, but all the metadata (data about the data) as well.

    Logical backups take more information than physical backups as they need to include the structure of the table – its relationship to other entities, index information, data types. The reason why much more metadata needs to be recorded is to make the logical backup useful in the recovery phase. Without all this metadata information and the actual data, logical backups lose their context and meaningfulness.

    The question then arises: "Why do a logical backup?" Logical backups are useful when we want to move data from one operating system to another operating system. Physical databases cannot do that as the file format is usually unique to each operating system. Further, the different database engines (MySQL, Oracle, IBM, Microsoft) are not physically compatible. Logical backups are very good for migrating data from one database to a completely different database and environment.

    Physical – Physical backups are a direct image copy of the physical database files on the disk. They are the fastest way to make a copy of the database. Using the operating system, a database administrator makes physical copies of the files and then usually stores this copy on a backup server or other media storage. In the event of a database failure, the physical copies can be restored to their original location, and the DBA can then replay all the transactions using the Crash Recovery log. (The Crash Recovery log is an area in memory that records every change we make in the database. This log is continually written to disk to avoid any loss of information. The Crash Recovery log's only purpose is to be used for recovery.)

To make a physical copy of the database we can either shut down the MySQL server or perform an open (sometimes called 'hot') backup.

- Backup mode – online or offline

  Online – Online backups mean that the users are still connected and are unaffected by the backup operations. There is no loss of availability of the database.

  Offline – Offline backups mean that the database server process is shut down while the physical copy of the file is made. No users can connect to the database or process any queries while a database is offline.

- Backup location – onsite or offsite

  Onsite – Onsite backups are stored on the same premises – but not the same machine – as the database.

  Offsite – Offsite backups are stored in a remote location (usually more than 160 km away from the primary site).

- Full backups and incremental backups

  We can back up all data in our database (a **full** backup) or take an **incremental** backup. An incremental backup only backs up the changes since the last backup.

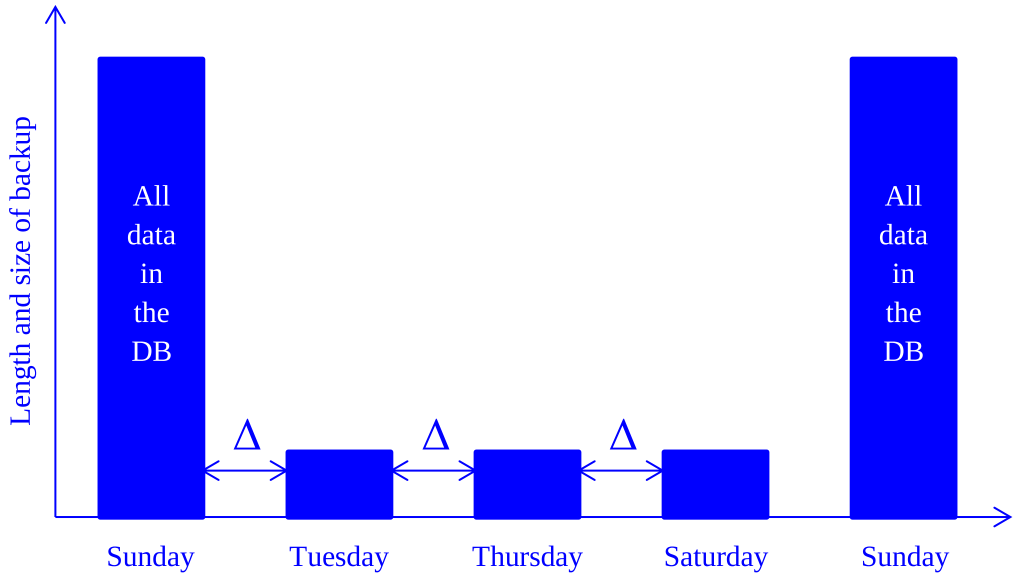


Figure S1: Full versus incremental backups. Full backups are larger in size and therefore take longer to complete.

In Figure S1, we take a full copy of all data in the database every Sunday. On Tuesday, Thursday and Saturday we only take the changes that have been made in the database since the last backup of any type (incremental or full). Tuesday’s backup will only backup data (rows, indexes, data dictionary information, etc.) that have been changed (including new data) since Sunday. Thursday's backup will only take changes made in the database since Tuesday, and Saturday’s backup will only take changes since Thursday. On the next Sunday, we once again back up all data stored in the database.

Incremental backups are smaller in size and shorter in duration. This has benefits if our database is doing batch processing overnight, as batch processing requires extensive machine resources to run. Incremental backups help for both recovery AND performance of the database.

Note: Within each backup, we can elect to back up only certain tables or schemas instead of backing up the entire database. This is known as a partial backup.

- How does database recovery work?

  Recovery of databases is nearly always in two phases: restore the backup then recover to the point of failure using the recovery log.

  1. The first phase is the restoration of the backup to the database server machine.

  2. The second phase is the recovery up until the point of the database failure (known as a full recovery). No committed data in the database should be lost.

Exercises continued:

2. Backup and recovery case study

ACME Manufacturing makes widgets in its factory. The factory runs 24 hours a day, 7 days a week in three shifts. The quietest shift is the Sunday night shift, which runs from midnight Sunday to 8am Monday. While ACME manufactures widgets, the database must run. This is ACME’s only widget factory.

The database administrator has implemented a backup policy that takes a full backup every Sunday at 3am during the night shift, and then an incremental backup on Tuesday, Thursday and Saturday mornings at 3am.

The backup strategy has determined that if there is a database failure, restoration of the database is time-critical. ACME must have the shortest outage time to restore and recover the database. This means the database must be restored quickly so that the manufacturing can continue. ACME must have the smallest elapsed time from the point of failure to the database being fully operational and useable.

a. Given the business requirements and the database administrator’s backup policy, what database backup type, mode and site would you recommend?

   You would recommend an *online, onsite, physical* backup. The online backup would mean there would be no interruption of operations at ACME. The physical backup is the preferred type of backup because it is the fastest to backup and restore.

While having an onsite backup creates a risk of a single point of failure, it is ACME's only widget factory, so if it was to be destroyed ACME does not have anywhere else to make widgets. An offsite backup provides little advantage.

b. Consider the Full and Incremental backup timeline in Figure 1. If the database suffered a media failure on Friday at 9:23am, how many backups would need to be restored?

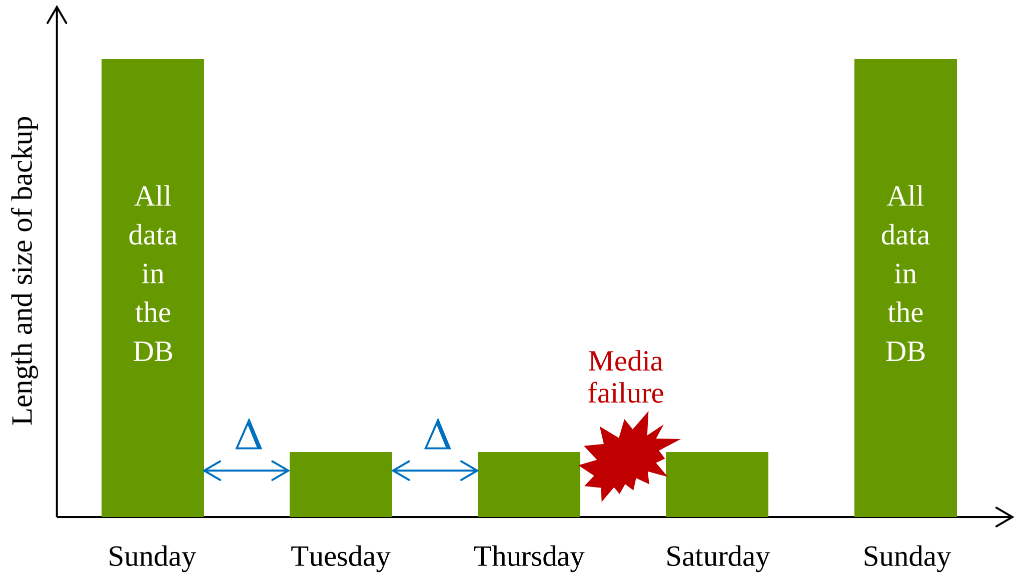


Figure 1. A timeline of full and incremental backups showing the media failure on Friday morning.

Three backups would need to be restored:

- The Sunday full backup
- The Tuesday incremental backup
- The Thursday incremental backup

The DBA would also replay the crash recovery logs from Thursday morning until Friday 9:23am.

c. Given the same failure, what would be the benefits and costs of changing the backup strategy to do full backups on Sunday, Tuesday, Thursday and Saturday mornings at 3am?

Benefits: In the restore and recovery phase, this strategy would reduce the time to fully restore the database from the last full backup and apply the crash recovery logs. This would mean ACME are going to be fully operational faster than if the Full and Incremental strategy was used.

Costs: More space would be required to back up the database as the backup will take everything and not just the changes. As there is more data to copy, the backup will take longer to complete. There is a risk that if we don't remove the backups from the database server, we could fill up the file system.

Key Concepts:

- What is a transaction?

A transaction is a logical unit of work that must either be entirely completed or aborted. A transaction usually corresponds to a single "action" that involves several changes to the database, for example, removing an employee and all their related data from the database, or an actual financial transaction that removes funds from one account and adds them to another. Transactions adhere to the ACID principles.

- ACID

The acronym ACID specifies four desirable properties of transactions. A DBMS that implements transactions that achieve these properties is said to be "ACID-compliant".

Transactions in an ACID-compliant database should be:

Atomic – Each transaction either entirely succeeds or entirely fails. If a failure occurs mid-transaction, the DBMS must discard (roll back) all of the transaction's changes up to that point.

Consistent – Upon completion, a transaction must respect data integrity rules (constraints) and leave the database in a consistent state.

Isolated – Changes made in one transaction cannot be seen from within other transactions. Transactions give the impression of being executed side-by-side simultaneously, although in practice, the need for data locking means that some transactions might be delayed by other transactions as they wait for locks to be released.

Durable – Once a transaction is committed, the inserts, updates and deletes carried out in that transaction persist permanently in the database.

- Concurrency

Concurrency is the ability to allow many users to connect to and work with the database simultaneously. It is possible for a database to allow concurrent access and still satisfy the ACID properties, but the isolation property creates particular challenges.

- The lost update problem

When two users attempt to update the same piece of data in the database at the same time, they might conflict with each other. One user changes the data value, and the other user does not see this update before writing their own update to that database. The first user's update is lost.

The classic example of the lost update problem is when two people are accessing the same bank account. Suppose my bank account contains $500. I am at an electronics store buying a $300 TV; at exactly the same time, my employer is paying me $120 in wages.

My transaction at the electronics store	**My employer's transaction**
Read account balance ($500)	
	Read account balance ($500)
Write account balance less $300 ($200)	
	Write account balance plus $120 ($620)

Instead of being $180 poorer, I am now $120 richer! This is not my fault – it's up to the bank to make sure the isolation property of ACID is satisfied and lost updates cannot happen.

Exercises continued:

3. Transactions

It's class registration day, when UniMelb students register in tutorial classes for the upcoming semester. In one particular subject, each tutorial class can fit a maximum of 24 students.

Eamonn and Jacqueline both wish to register in the Wednesday 10am tutorial class for this subject. This class already has 23 students enrolled – just one place remains.

Suppose the database contains tables like this:

TutorialClass (<u>SubjectCode</u> (FK), <u>TutorialNumber</u>, TotalEnrolments)

TutorialEnrolment (<u>SubjectCode</u> (FK), <u>TutorialNumber</u> (FK), <u>StudentNumber</u> (FK))

a. Describe how a lost update could occur in this database when Eamonn and Jacqueline try to simultaneously register in the Wednesday 10am tutorial.

Suppose Eamonn's enrolment request is received a split second before Jacqueline's. The server might execute the operations in this order (any order is correct so long as Jacqueline's *Read* comes before Eamonn's *Write*):

Eamonn	**Jacqueline**
Read TutorialClass.TotalEnrolments (23)	
	Read TutorialClass.TotalEnrolments (23)
Insert row into TutorialEnrolment	
	Insert row into TutorialEnrolment
	Write TutorialClass.TotalEnrolments (24)
Write TutorialClass.TotalEnrolments (24)	

Even though there are now 25 students enrolled in the class, the value of TotalEnrolments for this class is equal to 24. A lost update has occurred.

b. How could the lost update problem be avoided in this situation?

One solution is to enforce serial execution, where only one transaction is executed at a time. This will make the system very inefficient, but it will guarantee that the isolation property of ACID is satisfied.

A better solution is to use locking. When a transaction wishes to read the TotalEnrolments value for a class, it takes out a lock on that row of the TutorialClass table. This prevents other transactions from modifying that row. Once the transaction has finished writing to the row, it releases the lock. This approach is more efficient than serial execution, as a student's enrolment request only has to wait for the completion of other requests to enrol *in the same class*, instead of waiting for all other requests in the system.

It is also possible to use the other concurrency control methods outlined in the lecture, such as timestamps (if the timestamp of TotalEnrolments changes between when Eamonn reads it and when he is about to write it, Eamonn's transaction would abort and restart) or optimistic concurrency control (if TotalEnrolments is no longer equal to its original value when Eamonn is about to write it, Eamonn's transaction would abort and restart).

END OF TUTORIAL