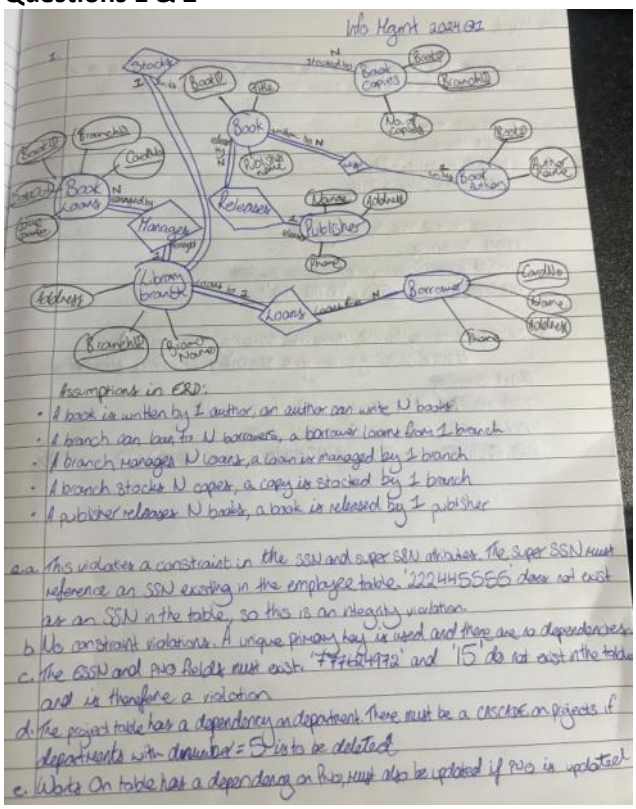
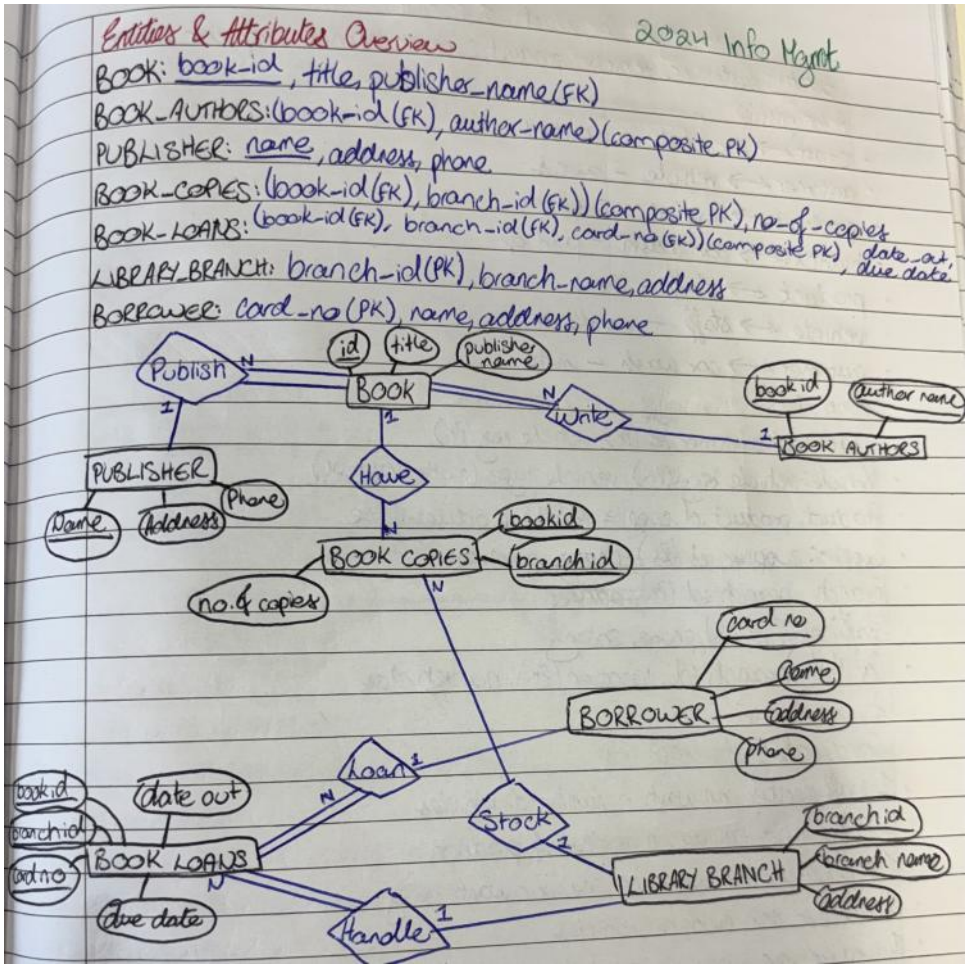
2024

Saturday, March 01, 2025 7:10 PM

Questions 1 & 2



Question 1 - Detailed



Entities:

- Book
- Book authors
- Publisher
- Book copies
- Book loans
- Library branch
- Borrower

Relationships:

- Book <---> Publisher - <publish>
- Book <---> Author - <write>
- Book <---> Book Copies - <have>
- Book Copies <---> Library Branch - <stock>
- Library Branch <---> Book Loans - <handle>
- Book Loans <---> Borrower - <loan>

Cardinalities:

- <Publish> -1 to n
- <Write> - 1 to n
- <Have> - 1 to n
- <Stock> - 1 to n
- <Handle> - 1 to n
- <Loan> - 1 to n

Attributes:

- Book – book id (PK) title, publisher name (FK)
- Book Authors – (book id (FK), author name) (composite PK)
- Publisher – name (PK), address, phone
- Book Copies – (book id (FK), branch id (FK)) (composite PK), no. Of copies
- Book Loans -(book id (FK), branch id (FK), card no. (FK)) (composite PK), date out, due date
- Library Branch – branch id (PK), branch name, address
- Borrower – card no (PK), name, address, phone

Question 2 – Detailed

a.

The employee table has a functional dependency on the supper ssn attribute. This entry cannot be inserted because the super ssn '222445555' does not exist in the database. This is a referential integrity violation. To prevent this violation from being applied to the database, a CONSTRAINT line can be added.

```
CONSTRAINT QuestionA FOREIGN KEY Super_ssn REFERENCES Employee(Ssn);
```

b.

No entity, referential or domain integrity violations.

Entity integrity: ensures that every entity has a unique and non-null primary key and prevents duplicate or null keys. This is to ensure every row is uniquely identifiable.

Referential integrity: Ensures that a foreign key in one table refers to a primary key in another table. This prevents orphaned records and pre

Domain integrity:

c.

None of the functional dependencies are met for this insertion. This is a referential integrity violation. To prevent this violation from being put into the database, we need foreign key constraints.

```
CONSTRAINT QuestionC FOREIGN KEY Essn REFERENCES Employee(Ssn);
CONSTRAINT AlsoQuestionC FOREIGN KEY Pno REFERENCES Project(Pnumber);
```

d.

There are records which reference Dnumber = 5 in the database. In order for this to be deleted without violating integrity constraints, there must be a defined `ON DELETE CASCADE`.

```
ALTER Employee
ADD CONSTRAINT DepartmentDelete
FOREIGN KEY (Dno) REFERENCES Department(Dnumber)
ON DELETE CASCADE;

ALTER Project
ADD CONSTRAINT DepartmentDelete
FOREIGN KEY (Dnum) REFERENCES Department(Dnumber)
ON DELETE CASCADE;

ALTER Dept_Locations
ADD CONSTRAINT DepartmentDelete
FOREIGN KEY (Dnumber) REFERENCES Department(Dnumber)
ON DELETE CASCADE;
```

e.
Works_on table has a dependency on Pno and must also be updated if Pnumber in the Project table is changed from 30 to 40:

```
ALTER TABLE Works_on
ADD CONSTRAINT ProjectUpdate
FOREIGN KEY (Pno) REFERENCES Project(Pnumber)
ON UPDATE CASCADE;
```

Question 3

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Consider the above relational database schema. Suppose that all the relations were created by (and hence are owned by) user X, who wants to grant the following privileges to user accounts A, B, C, D, and E. Write SQL statements to grant these privileges:

(a) Account A can retrieve or modify any relation except DEPENDENT and can grant any of these privileges to other users.

```
GRANT SELECT, INSERT, DELETE, UPDATE ON EMPLOYEE TO A WITH GRANT OPTION;
GRANT SELECT, INSERT, DELETE, UPDATE ON DEPARTMENT TO A WITH GRANT OPTION;
GRANT SELECT, INSERT, DELETE, UPDATE ON DEPT_LOCATIONS TO A WITH GRANT OPTION;
GRANT SELECT, INSERT, DELETE, UPDATE ON PROJECT TO A WITH GRANT OPTION;
GRANT SELECT, INSERT, DELETE, UPDATE ON WORKS_ON TO A WITH GRANT OPTION;
```

(b) Account B can retrieve all the attributes of EMPLOYEE and DEPARTMENT except for SALARY, MGRSSN, and MGRSTARTDATE.

Need to create a view that excludes the restricted columns then grant to account B

```
CREATE VIEW AccountBEmployeeView AS
SELECT Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Super_snn, Dno
```

FROM EMPLOYEE;

GRANT SELECT ON AccountBEmployeeView TO B;

CREATE VIEW AccountBDeptView AS
SELECT Dname, Dnumber
FROM DEPARTMENT;

GRANT SELECT ON AccountBDeptView TO B;

(c) Account C can retrieve or modify WORKS_ON but can only retrieve the FNAME, MINIT, LNAME, SSN attributes of EMPLOYEE and the PNAME, PNUMBER attributes of PROJECT.
GRANT SELECT, INSERT, DELETE, UPDATE ON WORKS_ON TO C;

CREATE VIEW AccountCEmployeeView AS
SELECT Fname, Minit, Lname, Ssn
FROM EMPLOYEE;

GRANT SELECT ON AccountCEmployeeView TO C;

CREATE VIEW AccountCProjectView AS
SELECT Pname, Pnumber
FROM PROJECT;

GRANT SELECT ON AccountCProjectView TO C;

(d) Account D can retrieve any attribute of EMPLOYEE or DEPENDENT and can modify DEPENDENT.
GRANT SELECT ON EMPLOYEE TO D;
GRANT SELECT, INSERT, DELETE, UPDATE ON DEPENDENT TO D;

(e) Account E can retrieve any attribute of EMPLOYEE but only for EMPLOYEE tuples that have DNO = 3.
CREATE VIEW AccountEEmployeeView AS
SELECT * FROM EMPLOYEE WHERE Dno = 3;

GRANT SELECT ON AccountEEmployeeView TO E;

Question 4
What is GDPR? Include in your answer what jurisdiction GDPR applies to and what legislation it now replaces.
GDPR governs how personal data is collected, processed and protected.
It aims to strengthen individual's rights regarding their personal data and ensure greater transparency and accountability from organisations handling data.
GDPR applies to all organisations in the EU who process data.
It applies to companies outside the EU if they process the data relating to individuals residing in the EU, regardless of where the company is based.
GDPR replaced the 1995 EU Data Protection Directive on May 25 2018.

List the GDPR principles.
1. Lawfulness, fairness and transparency.
2. Purpose limitation.
3. Data minimisation.
4. Accuracy
5. Storage limitation.
6. Integrity and confidentiality.
7. Accountability.

What is personal data?
Under GDPR, personal data refers to any information that relates to an identifiable data subject.
Direct identifiers: name, address, phone number, email, etc.
Indirect identifiers: IP addresses, cookies, location services, etc.
Sensitive data: health records, biometrics, ethnic background, religious views, etc.

Question 4 – Detailed
General Data Protection regulation (GDPR) is an applicable law in all EU member states which outlines laws regarding the processing and controlling of personal data in the EU. All people, companies, bodies which operate in the EU are subject to the laws of GDPR. It came into effect on May 25th 2018k, replacing the 1995 EU Data Directive.

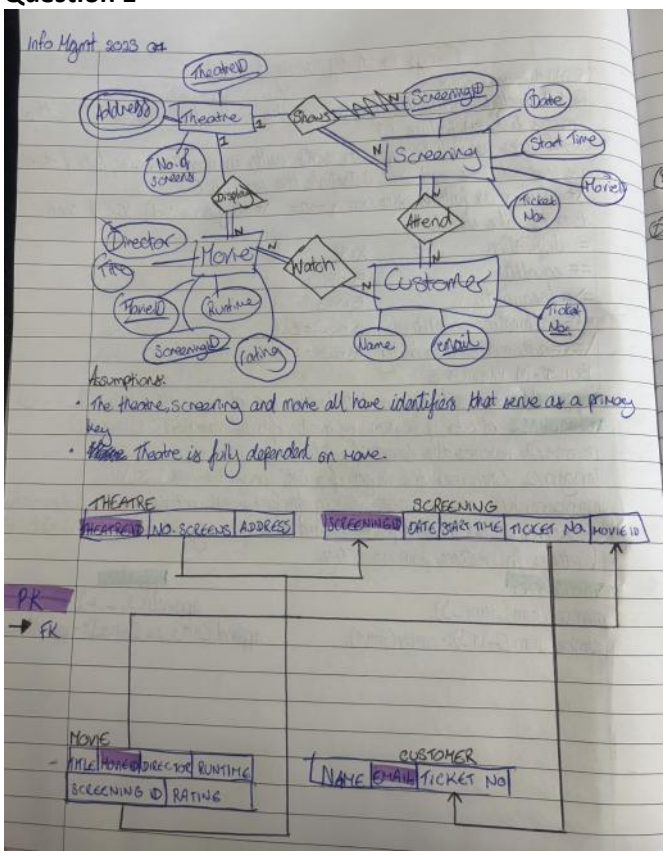
1. Lawfulness, fairness, transparency
2. Data minimization
3. Purpose limitation
4. Storage limitation
5. Accuracy
6. Integrity and confidentiality
7. Accountability

Personal data is any information that a data controller/processor uses which relates to an identified/identifiable data subject. Personal data can be direct, indirect or sensitive. Direct identifiers are full names, addresses, etc. Indirect data are things like medical records, IP addresses, location services, etc. Sensitive data is things like racial or ethnic background, political opinions, etc.

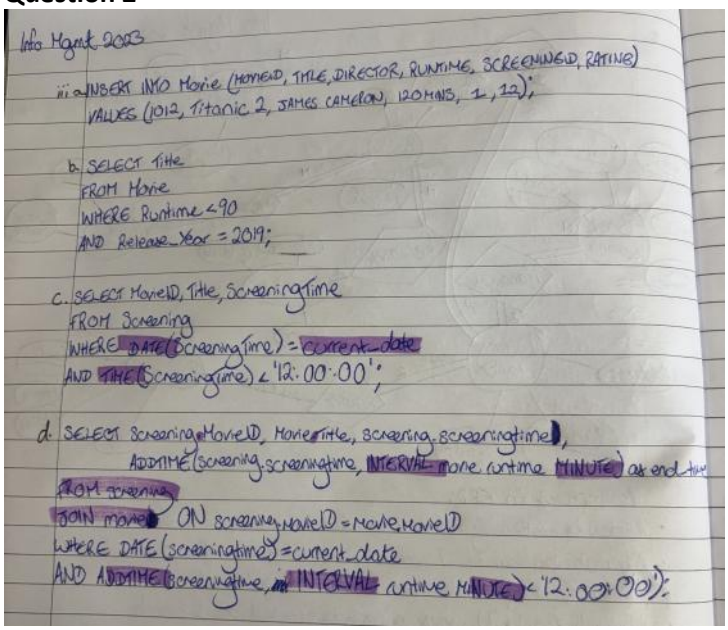
2023

Saturday, March 01, 2025 8:10 PM

Question 1



Question 2



Describe three types of integrity constraints that can be applied to relational databases.

1. Entity Integrity: ensures that each table has a unique and non-null primary key. Prevents duplicate or null values in the primary key column, ensuring every row is uniquely identifiable.
2. Referential Integrity: ensures that a foreign key in one table refers to a primary key in another table. Prevents orphaned records, records in a child table with a non-existent parent table. E.g. if a department number in the Employee table references the Department table, that department must exist.

3. Domain Integrity: ensures that each column contains only valid values based on predefined rules. Constraints like CHECK, NOT NULL, UNIQUE enforce domain integrity. E.g. restricting the values of Salary to be above zero.

Describe how referential integrity can be specified in SQL giving an example of your choice with a clear explanation of why referential integrity constraints are appropriate.
Referential integrity ensures that foreign key values in a child table always match valid values in a parent table.

```
CREATE TABLE Department (
    Dnumber INT PRIMARY KEY,
    Dname VARCHAR(100) UNIQUE
);

CREATE TABLE Employee(
    Ssn INT PRIMARY KEY,
    Fname VARCHAR(50),
    Lname VARCHAR(50),
    Dno INT,
    FOREIGN KEY (Dno) REFERENCES Department(Dnumber) ON DELETE CASCADE
);
```

Dno in the Employee table references Dnumber in the Department table.
If a department is deleted, ON DELETE CASCADE ensures that all employees belonging to that department are automatically deleted to prevent orphaned records.
Referential integrity constraints prevents inconsistent data, maintains logical relationship between tables and ensures database reliability

Consider the below relations in a database that keeps track of stock levels in a music and movie store. Write an SQL statement that creates an assertion called maximum_inventory that prevents insertions into either table violating the following condition: the sum of the sale_price of all movies and music combined must not exceed 1,000,000 euro.

MOVIE

ID	Title	Director	Genre	Cost_Price	Sale_Price	...

MUSIC

ID	Title	Artist	Genre	Cost_Price	Sale_Price	...

```
CREATE ASSERTION maximum_inventory
CHECK(
    (SELECT COALESE(SUM(Sale_Price),0) FROM MOVIE) +
    (SELECT COALESE(Sum(Sale_Price,0) FROM MUSIC)
    <=1000000
);
```

The sum of bot totals must be /< €1,000,000

Explain any differences you are aware of between assertions and triggers in relational databases.
Assertions are global constraints but are not widely supported.
Triggers are more widely used and flexible as they can perform additional actions e.g. automatic updates.
If cross-table constraints are needed, a trigger on each affected table is used as an alternative to assertions.

Assertions enforce database-wide rules and conditions.
Triggers execute automatically before/after certain operations (INSERT,UPDATE,DELETE).
Triggers are applied at the table-level and triggered by specific events.
Triggers may require multiple triggers for multiple tables.
Assertions prevent invalid transactions by rejecting operations that violate the condition.

Question 3
Explain what is a transaction and how transactions can fail.
A transaction in a database is a sequence of operations performed as a single unit of work.
A transaction is either committed or rolled-back to maintain database integrity.

BEGIN TRANSACTION;

UPDATE Accounts SET Balance = Balance – 100 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;

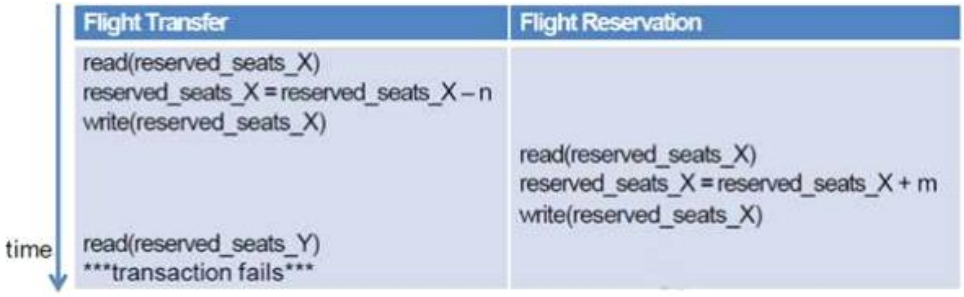
COMMIT;
Transferring €100 from account 1 into account 2 should happen successfully or not at all; NOT partially.

1. Transactions can fail due to a system crash, software failure, power outage, etc.
2. Transactions can fail due to deadlock where two or more transactions are waiting indefinitely for resources locked by each other.
3. Transactions can fail due to concurrency issues e.g. two customers booking the same aeroplane seat at the same time.
4. Transactions can fail due to a constraint violation. If a transaction violates an integrity rule, it will fail.
5. Transactions can be explicitly rolled-back due to errors in the application logic. E.g. a user cancels an order before final payment.

What are the ACID properties? Describe in detail.
The ACID properties ensures that transactions in a database are:
Atomicity, Consistency, Isolation, Durability
A: a transaction must be fully completed or fully undone. If any part of the transaction fails, the transaction is fully rolled back.
C: a transaction must leave the database in a valid state. Data must comply with integrity constraints before and after the transaction.
I: transactions must not interfere with each other and should execute as if they were done sequentially. Two users booking the same seat at the same time should not both be able to purchase it.
D: once a transaction is committed, its changes must be permanent even if a system crash occurs.

Property	Definition	Example
Atomicity	"All or nothing" – A transaction is fully completed or fully undone.	If a payment process fails, the money is not deducted.
Consistency	Ensures database integrity before and after a transaction.	Total bank balance should remain valid after a transfer.
Isolation	Prevents transactions from interfering with each other.	Two customers should not book the same seat at once.
Durability	Once committed, changes are permanent, even after failures.	A placed order remains in the system even if the server crashes.

In the below example, a Flight transfer transaction (between flights X and Y) and a Flight reservation (flight X only) transaction are being carried out simultaneously on the same database. The flight transfer transaction fails after reading data item reserved_seats_Y. Explain the type of error that will occur in this case.



This causes a deadlock error in the transaction.

Flight Transfer reads resevered_seats_X, modifies it and writes it back to the database.

Flight Reservation reads reserved_seats_X, modifies it and writes it back to the database.

Flight transfer tries to read reserved_seats_Y while Flight Reservation still holds the lock on reserved_seats_X.

If Flight Reservation tries to access reserved_seats_Y, both transactions end up waiting for the other to release locks.

Question 4

A prominent telecommunications company, Telemate, based in Ireland, is eager to get its latest deal out to its valued customers. An email list is set up including the email addresses of all existing customers. The offer that Telemate is putting together will be highly cost-effective for customers and the marketing manager, Helen, would like this information to reach as many customers as possible, as quickly as possible. An email is composed that highlights the most important parts of the offer, including a deal that will allow contract customers to own the latest iPhone at 50% of the usual contract price. Helen instructs Tom to send out the email to all existing customers on October 6th 2020. One customer, John, receives the email along with several other emails that John wastes time deleting, he is not interested in buying an iPhone. John would like to avoid receiving marketing emails altogether and never agrees to receiving these kinds of emails from companies.

Describe in detail what John's rights are in terms of GDPR and what action John should initially take.

1. Right to Object: John can object to the processing of his personal data. Once John objects, Telemate must stop sending him marketing emails immediately.
2. Right to withdraw Consent: Telemate are already in violation of GDPR as John never agreed to marketing emails. But if John had previously agreed, he has the right to remove his consent at any time.
3. Right to Erasure/Right to be Forgotten: John can request that Telemate delete his personal data from their marketing database to prevent further emails

John should choose the Unsubscribe option required by GDPR that should be on the marketing email.
John should email Telemate's Data Protection Officer or customer support requesting to opt out of all marketing emails in the future.
John should formally request that his email and personal details be removed from the marketing list under the Right to Erasure.

Who in Telemate is responsible for the potential violation of GDPR?

1. Helen: Helen instructed the mass email campaign without verifying whether or not customers had given their explicit consent to receive marketing emails.
2. Tom: Tom executed the campaign and should have flagged the GDPR violation with his management instead of proceeding.
3. Telemate's Data Protection Officer: Telemate's DPO is responsible for ensuring compliance.
4. Telemate as a company: The company itself is liable for any non-compliance and could face fines.

Despite John carrying out the action you described in (i), John was not satisfied with the result and in fact received a second similar email about a different offer 6 weeks later. What action can John now take and what are the possible repercussions for Telemate?

File a complaint with the Data Protection Commission: John can lodge a formal complaint. The DPC will investigate Telemate's practices and enforce penalties if a violation is confirmed.

Repercussions for Telemate: GDPR violations can result in fines of €20 Million or 4% of annual global turnover, whichever is higher. John could take legal action against Telemate for misuse of his personal data. A data breach could affect Telemate's reputation and customer trust.

Rights under GDPR:

1. Right to be forgotten – John can request at any time that Telemate stop using his personal data
2. Right to object – John can refuse to allow Telemate to send him marketing emails if he does not wish to receive them
3. Right to be informed – John has a right to know for what purpose Telemate is using his personal data and what data Telemate is processing.
4. Right to rectification – John has the right to correct Telemate on the personal data which they hold regarding him
5. Rights on automated processing – John has the right to choose to not allow automated processes to be carried out on his personal data I.e. if John only wants his data to be processed by a real person, he has this right.
6. Right to access – John has the right to request from Telemate the personal data of his which they are processing.
7. Right of portability – John has a right to obtain his personal data from the data controller in a format which makes it easier to reuse.
8. Right to withdraw consent – At any time, John has the right to withdraw consent to processing that he may have given to Telemate.

2021

Wednesday, April 16, 2025 9:33 PM

The ReallyGoodRestaurant has decided to open a small shop beside the restaurant to sell high quality artisan cooking ingredients. The restaurant uses several suppliers to provide good ingredients. Each supplier supplies an ingredient in only one unit size e.g. supplier A supplies sea salt in bags of 500 grams and supplier B supplies sea salt in 250 gram bags. Each supplier charges a specific price for an ingredient in that unit size. However different suppliers can provide the same ingredient in different unit sizes. Also to encourage the sale of these ingredients, the restaurant provides a text description of the recipes used in the restaurant and indicates which ingredients from each recipe can be bought from the shop. Each recipe has been designed by one or more chefs in the restaurant. The ReallyGoodRestaurant needs to maintain a database capable of storing information about ingredients, the suppliers of those ingredients, the unit size that each supplier supplies a particular ingredient, a text description of each recipe, the name(s) of the chef(s) who designed each recipe, an indication of which ingredients are used in which recipes, the cost (to the restaurant) of ingredients in a particular unit sizes from suppliers, and the price for which the shop sells each ingredient (in a particular unit size).

Develop an Entity Relational (ER) Model for the above database, stating any assumptions you make.

Boxes:
- Supplier
- Ingredient
- Recipe
- Chef

Diamonds:
- Supplier <sell> Ingredient
- Ingredient <use> Recipe
- Recipe <create> Chef

Supplier Attributes:
- SupplierID
- SupplierName

Ingredient Attributes:
- IngredientID
- IngredientName
- NetWeight

Recipe Attributes:
- RecipeID
- RecipeTitle
- TextualDescription

Chef Attributes:
- ChefID
- ChefName

<sell> Attributes:
- IngredientUnitSize
- IngredientSellingPrice
- IngredientBuyingCost

<use> Attributes:
- QuantityUsed

<create> Attributes:
- CreationDate

Using the appropriate mapping techniques, map the ER Model to a Relational Model and show the functional dependency between attributes within each table. In your answer, identify the Primary and Foreign keys for each relation.

Table	*PK*	*FK*	*Functional Dependencies*
Supplier	SupplierID		SupplierID -> SupplierName
Ingredient	IngredientID		IngredientID -> IngredientName, NetWeight
Recipe	RecipeID		RecipeID -> RecipeTitle, TextualDescription
Chef	ChefID		ChefID -> ChefName
SupplierIngredient	SupplierID, IngredientID	SupplierID -> Supplier(SupplierID) IngredientID -> Ingredient(IngredientID)	(SupplierID,IngredientID) -> UnitSize, SellingPrice, BuyingCost
IngredientRecipe	IngredientID, RecipeID	IngredientID -> Ingredient(IngredientID) RecipeID -> Recipe(RecipeID)	(IngredientID,RecipeID) -> QuantityUsed
RecipeChef	RecipeID, ChefID	RecipeID -> Recipe(RecipeID) ChefID -> Chef(ChefID)	(RecipeID,ChefID) -> CreationDate

Give the SQL command(s) for the following queries:

Retrieve a list of ingredients sold by the shop for the recipe 'Baked Lasagne'

1. *SELECT DISTINCT i.IngredientID*
2. *FROM Ingredient i*
3. *JOIN IngredientRecipe ir ON i.IngredientID = ir.IngredientID*
4. *JOIN Recipe r ON ir.RecipeID = r.RecipeID*
5. *JOIN SupplierIngredient si ON i.IngredientID = si.IngredientID*
6. *WHERE r.RecipeTitle = 'Baked Lasagne';*

1. Retrieves the unique IDs or ingredients, ensures there are no duplicates returned even if an ingredient is sold by more than one supplier or appears more than once in intermediate joins.
2. Begins the query from the Ingredients table.
3. Links each ingredient (i) to the IngredientRecipe table. Tells us which ingredients are used on which recipes.
4. Joins the previous result to the Recipe table. Now we can filter by specific attribute e.g. recipe name.
5. Joins the previous result to the SupplierIngredient table to ensure that the ingredient is actually sold by the shop.
6. Filters by recipe title for the name 'Baked Lasagne'

One of the existing suppliers (named 'Honest Henry') has a new ingredient (roasted chestnuts in units size of 400 grams) to be sold in the shop. The shop buys this ingredient €3 per 400 gram bag and the shop sells it for €8 per (400 gram bag). Give the SQL commands needed to enter this new ingredient.

INSERT INTO Ingredient (IngredientID, IngredientName, NetWeight)
VALUES(200, 'Roasted Chestnuts', 400);

INSERT INTO SupplierIngredient(SupplierID, IngredientID, UnitSize, BuyingCost, SellingPrice)
VALUES(1,200,400,3.00,8.00);

Create a view of all the ingredients of supplier 'Honest Henry' with the unit size he supplies and the price he charges for that unit size.

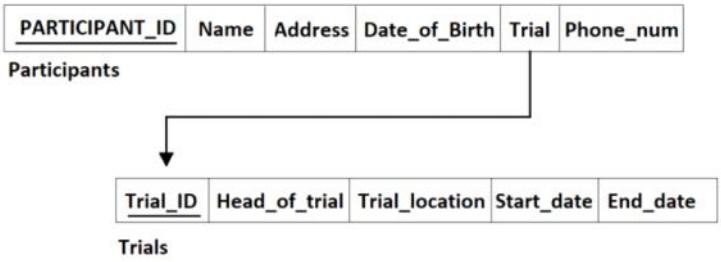
1. *CREATE VIEW HonestHenryIngredients AS*
2. *SELECT*
   a. *i.IngredientName*
   b. *si.UnitSize*
   c. *si.BuyingCost*
3. *FROM Supplier s*
4. *JOIN SupplierIngredient si ON s.SupplierID = si.SupplierID*
5. *JOIN Ingredient i ON si.IngredientID = i.IngredientID*
6. *WHERE s.SupplierName = 'Honest Henry';*

1. Creates view with name HonestHenryIngredients.
2. Specifies the columns that will be displayed in the view: IngredientName from the Ingredient table, UnitSize from the SupplierIngredient table, BuyingCost from the SupplierIngredient table.
3. Specifies the table to start pulling the information from.
4. Joins the Supplier table with the SupplierIngredient table to connect each supplier with the ingredients they supply.
5. Joins the above result with Ingredient to get the name of each ingredient in the result.
6. Filters by the supplier name being Honest Henry

Suppose the restaurant wishes to increase the prices for all ingredients it sells in the shop by 8%. Give the SQL command(s) required to make this happen.

UPDATE SupplierIngredient
*SET SellingPrice = SellingPrice * 1.08;*

A database manager is designing a relational database to store information about research laboratory experiments involving Covid-19 vaccine trials. Below is part of the database design for information stored about vaccine trials:



When setting up human trials for experiments, some human participants will be added to a trial but can later become ineligible and need to be removed from the trial. Participants only take part in a single trial but this is not explicitly enforced within the schema. So, if someone is a participant in a trial for vaccine A, that is the only trial they will participate in.

If a human participant is deleted from a trial, then all of his/her information should be deleted. Say what kind of constraint this applies to, security or integrity.

This is an integrity constraint. Integrity constraints are related to the data itself. There should be a cascading delete when a person is removed from the trial, allowing all their personal information to be removed from the other tables in the database.
Security constraints relate to who can access or modify data e.g. UPDATE, DELETE, MODIFY, SELECT

Below is a list of potential operations that could be carried out on the database. For each of the below examples, say whether or not the operation violates integrity constraints and if so, the kind of integrity constraint that is violated and why it is violated:

A new trial is started and information needs to be stored about that trial. Because we don't yet know the ID number of the trial at this point, an insert is carried out on the Trials table to insert the information about the new Trial, leaving Trial_ID null for now.

This violates entity integrity. You cannot have a null primary key without violating entity integrity.

Another trial begins, this time they do not yet have an estimated end date for the trial, so a tuple is inserted into the Trials table with a single attribute set to null, the End_date.

This violates domain integrity. Domain integrity ensures that all columns have valid values based on predefined rules

A trial is deleted from the Trials table with nothing removed from any other tables.
This immediately violates referential integrity. Without a cascading delete, there will be orphaned records that are referencing information that has been deleted from the database.

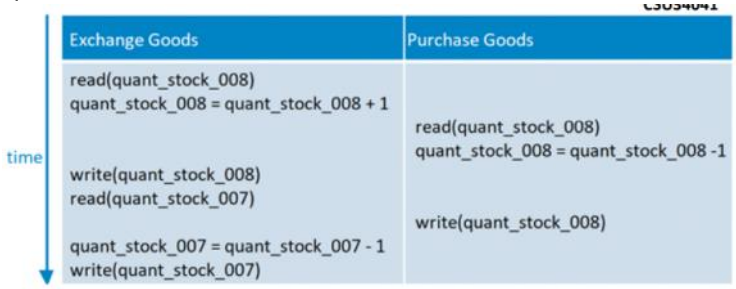
Provide the SQL command to add a constraint that enforces the start date of all human trials to be after 28th February 2019.

```
ALTER TABLE Trials
ADD CONSTRAINT TrialStartDate
CHECK (Start_Date > DATE '2019-02-28');
```

Provide the SQL to avoid a referential integrity violation that would result from a Trial being deleted from the Trials table.

```
CREATE TABLE Participants(
        ParticipantID INT PRIMARY KEY,
        Name Varchar(100),
        TrialID INT,
        FOREIGN KEY (TrialID) REFERENCES Trials(TrialID)
                ON DELETE CASCADE
);
```

Question 3 – Transactions



Provide the schedule of operations depicted in the diagram, where Exchange Goods is Transaction1 and Purchase Goods is Transaction2, and a list of any potential conflicts.
Schedule of operations:

Order	Transaction	Operation
1	T1	Read(quant_stock_008)
2	T1	Quant_stock_008 = quant_stock_008 +1
3	T2	Read(quant_stock_008)
4	T2	Quant_stock_008 = quant_stock_008 -1
5	T1	Write(quant_stock_008)
6	T1	Read(quant_stock_007)
7	T2	Write(quant_stock_008)
8	T1	Quant_stock_007= quant_stock_007 -1
9	T1	Write(quant_stock_007)

There are potential conflicts as both transactions are accessing the quant_stock_008 attribute at the same time and both making changes.
T2 is making changes to the attribute before T1 has written its own changes.
For example if quant_stock_008 is 100 and both transactions read this, T1 updates it to 101 and T2 updates it to 99, then T1 writes 101, then T2 writes 99.

Provide an alternative schedule that fixes any issues with the above schedule and describe any advantages or disadvantages of the approach you chose.

Order	Transaction	Operation
1	T1	Read(quant_stock_008)
2	T1	Quant_stock_008 = quant_stock_008 +1
3	T1	Write(quant_stock_008)
4	T2	Read(quant_stock_008)
5	T2	Quant_stock_008 = quant_stock_008 -1
6	T2	Write(quant_stock_008)
7	T1	Read(quant_stock_007)
8	T1	Quant_stock_007= quant_stock_007 -1
9	T1	Write(quant_stock_007)

In this new schedule, T2 waits until the changes from T1 have been written to the database before T2 attempts to read the data and make its own changes.
This way there will be no conflicts, however operations will be slower as the two transactions are no longer operating concurrently.

Add locking to the schedule and explain what will occur when the wait-die protocol is applied, making use of a wait-for graph in your explanation.

In the wait-and-die protocol, let T1 be the older transaction and let T2 be the newer transaction.
Locks are applied to the transactions' shared resources e.g. quant_stock_008, quant_stock_007, etc.
If T2 wants to access a resource that is currently locked by T1, T2 dies (rolls back the transaction and starts again later).
WFG:
T2 -> T1 (directed edge from T2 to T1, as T2 is waiting on the resource that T1 has.
After T2 dies, there is no edge in the WFG.

2020

Thursday, April 17, 2025 12:11 PM

A newspaper publisher wishes to keep track of the articles to be printed within each edition of the newspaper, the authors of these articles and a set of descriptive key words (topics) which characterise the articles. For each author the newspaper maintains the authors name, the (topical) areas within which he/she has expertise, the address and phone number. The publisher publishes a different edition of the newspaper each day. For each article, the publisher maintains the title of the article, its author(s), the number of words in the story and the topical area(s) which are covered in the article. An article which is not yet published, will not have an edition of the newspaper associated with it. Each article can actually be published in several editions of the newspaper and an edition is published on a single day. Each edition of the newspaper must have a single edition editor who chooses the articles for that edition.

Entities:

- Articles
- Edition
- Author
- Editor

Relationships:

- Author <WRITE> Article
- Article <APPEAR> Edition
- Edition <CHOOSES> Article, Editor

Entity Attributes:

Article:

- ArticleID
- Title
- No. Of words

Edition:

- EditionID
- Date – day, month, year

Author:

- AuthorID
- Name
- Address
- Phone Number

Editor:

- EditorID
- EditorName

Relationship Attributes:

Author <WRITE> Article:

- Topic of Expertise

Article <APPEAR> Edition:

- No. Of Editions

Give the SQL command(s) for the following queries:

a. Insert a new article written by 'John Smith' on the topic 'Boris Johnson loses election again'. You may suggest appropriate attribute values for the article. You may assume that John Smith is already an author of other articles and has an unique identification number (e.g. AuthorID) of 7777.

```
INSERT INTO Articles (ArticleID, Title, NoOfWords)
VALUES(100, 'Boris Johnson loses election again',  2000);
INSERT INTO ArticleAuthor(ArticleID, AuthorID, Topic)
VALUES (100, 7777, ' Boris Johnson loses election again');
```

b. Find out the topics of all the articles which were published on 2091-11-11 which were more that 100 words long.

[4 Marks]

```
SELECT aa.Topic
FROM ArticleAuthor aa
JOIN Articles a ON aa.ArticleID = a.ArticleID
JOIN ArticleEdition ae ON aa.ArticleID = ae.ArticleID
JOIN Editions e ON ae.EditionID = e.EditionID
WHERE a.NoOfWords > 100 AND e.Day = 11 AND e.Month = 11 AND e.Year = 2091;
```

c. Create a view in the database of all the authors who have expertise about crime and the titles of the articles that they have written.

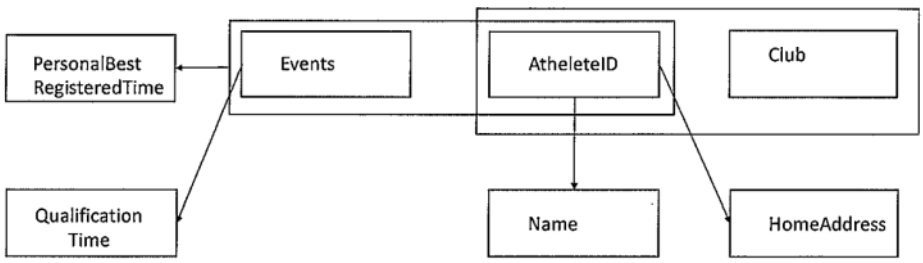
[3 Marks]

```
CREATE VIEW CrimeArticles AS
SELECT
     ar.Title
     aa.Topic
     au.Name
FROM Articles ar
JOIN ArticlesAuthor ON ar.ArticleID = aa.ArticleID
JOIN Author au ON aa.AuthorID = au.AuthorID
WHERE aa.Topic - "Crime";
```

d. Change the author name of the author whose AuthorID is 8888 to the name 'Donal Daly'

[2 Marks]

```
UPDATE Author
SET AuthorName = "Donal Day"
WHERE AuthorID = 8888;
```



(i) Identify the primary keys, foreign keys and other attributes of EACH of the tables in the above diagram.

[12 Marks]

Identifying primary keys, foreign keys and attributes from a functional dependency diagram

Primary Keys:

- AthleteID is the primary key of the Athlete table
- (AthleteID, Events) is the primary key of the composite entity that defines an athlete's time for

a given event
- (AthleteID, Club) is the primary key of the composite entity that defines what club an athlete belongs to

Foreign Keys:
- AthleteID from the Athletes table is an FK of the Athletes/Events composite
- AthleteID from the Athletes table is an FK of the Athlete/Club composite

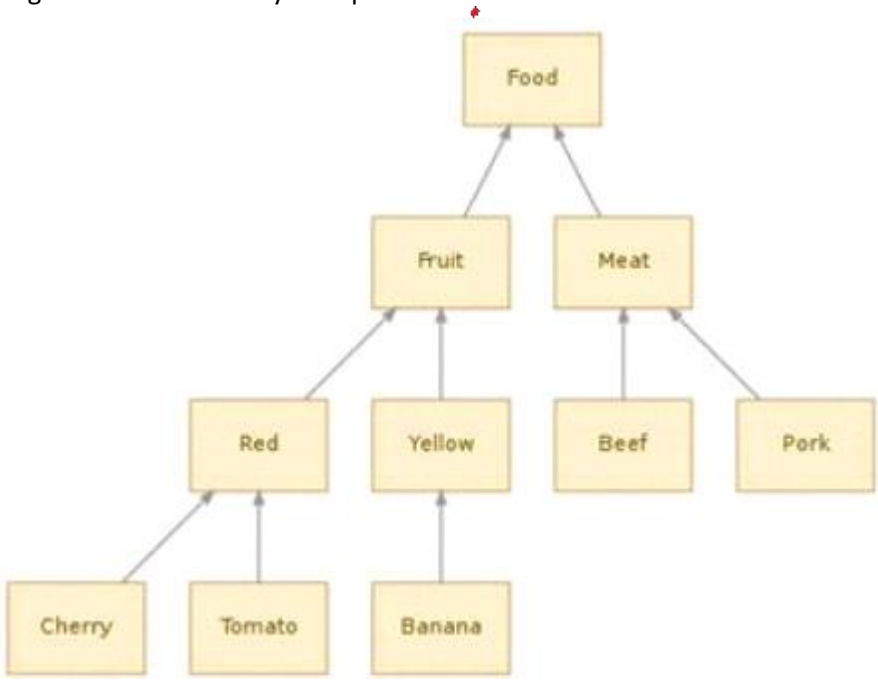
Attributes:
- Personal best is an attribute of the Athlete/Event composite
- Qualification time is an attribute of Events
- Name is an attribute of athlete
- Home address is an attribute of athlete

1st Normal Form: all attributes contain atomic values. E.g. a multi-valued attribute like Home Address violates first normal form.
2nd Normal Form: is in 1st normal form, and all attributes are dependent on their entire primary key, not just partially. E.g. if there is a composite entity X that has a composite primary key (Y,Z) then the attributes of X must be functionally dependent on both Y and Z, not one or the other.
3rd Normal Form: is in 2nd normal form and has no transient dependencies. For example, if Attribute C is functionally dependent on Attribute B, but Attribute B is functionally dependent on Attribute A, then Attribute C is transiently functionally dependent on Attribute A
Boyce-Codd Normal Form: all attributes are functionally dependent on the key, the whole key and nothing but the key

Database Models

Wednesday, January 29, 2025 4:08 PM

Hierarchal Database
Organised hierarchically with parent and child nodes

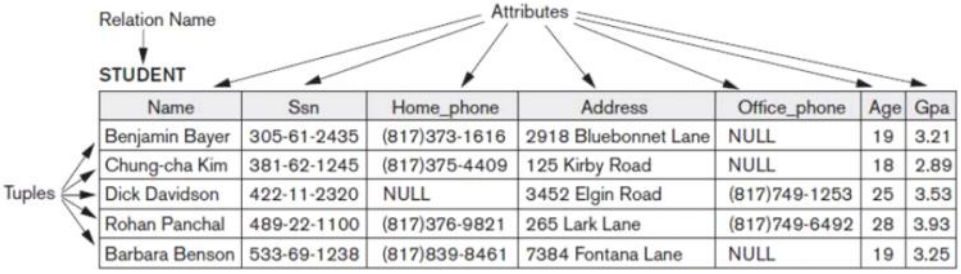


Network Database
Also have a hierarchical structure
Uses “members” and “owners” rather than “parents” and “children”.
Each member can have more than one owner

Object-Oriented Database
Attempts to Model Data Storage in a similar fashion to application programs
Graph Database
Uses a graph structure with: – Nodes – Edges – Properties
Graph databases treat the relationship between things as equally important to the things themselves.

Relational Database

More flexible than either the hierarchical or network database models.
Database represented as a collection of mathematical relations
The table, or relation, is the basic storage structure of a Relational Database.
Each row, or tuple, in a table represents a collection of related values
Each column, or attribute, contains values of the same data type



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Domain

The data type describing the values that can appear in each column is represented by a domain of possible values
mobile_phone_number: The set of 10 digit phone numbers valid in Ireland
PPS_number: 9 characters in length. 7 numeric characters in positions 1 to 7, followed by 1 alphabetic check character in position 8, and either a space or the letter “W” in position 9

Characteristics of Relations

Ordering of values within a tuple
– Each tuple t is an ordered list of n values
– Order can change as long as correspondence between attributes and values is maintained
Values in tuples – Each value in a tuple is atomic e.g. student age
Multivalued attributes For example: College Degree Must be represented by separate relations
Composite attributes For example: Address
NULL values

Relational model constraints:

1. Explicit Constraint (Enforced by the Database)

An **explicit constraint** is a rule **directly enforced** by the database schema to ensure data integrity.
These constraints prevent invalid data from being entered at the database level.

Example: UNIQUE Constraint (Explicit Constraint)

```sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,  -- Ensures uniqueness and non-null values
    Name VARCHAR(100) NOT NULL,  -- Prevents NULL values
    Email VARCHAR(255) UNIQUE  -- Ensures no duplicate emails
);
```

- **PRIMARY KEY** ensures each `CustomerID` is unique.
- **NOT NULL** ensures the `Name` field is always filled.
- **UNIQUE** ensures no two customers have the same email.

2. Application Constraint (Enforced by the Application Code)

An **application constraint** is a rule **implemented in the application layer** (e.g., in Java, Python, or JavaScript) instead of the database. This means invalid data could still be inserted if the database is accessed directly.

Example: Business Rule Enforced in an Application (Application Constraint)

```python
def create_customer(name, email):
    if not name:
        raise ValueError("Customer name is required.")  # Enforcing NOT NULL in code

    if "@" not in email:
        raise ValueError("Invalid email format.")  # Additional validation not in DB

    query = "INSERT INTO Customers (Name, Email) VALUES (?, ?)"
    db.execute(query, (name, email))
```

- The function checks if `name` is empty before inserting (similar to `NOT NULL` in SQL).
- It ensures the email format is correct (which the database **does not enforce by default**).

Architecture

Monday, February 17, 2025 11:35 AM

3-Level DBMS Architecture

External Level:

- The highest level of abstraction that deals with the user's view of the database
- Most users and applications do not require access to the entire data stored in the database
- External Schemas (or User Views) describe a part of the database for a particular group of users or applications
- Powerful and flexible security mechanism, as parts of the database are hidden from certain users
- User is not aware of the existence of any attributes that are missing from the view

Conceptual Level:

- Deals with the logical structure of the entire database
- Describes what data is stored in the database and the relationships among the data without any concern for the physical implementation
- Overall view of the database and includes all the information that is going to be represented in the database

Internal Level:

- The lowest level of data abstraction
- Internal Schema describes how the data is physically stored and organized on the storage medium
- Various aspects are considered to achieve optimal runtime performance and storage space utilization, including: storage space allocation techniques, access paths such as indexes, data compression and encryption techniques

DDL Compiler: processes schema definitions and stores them in a catalogue
Catalogue: contains, file name and size, names and data type of data items, storage details, constraints
Stored Data Manager: controls access to DBMS information on disk including buffer management
DBMS Users: casual users, application programmers, parametric users, DBA staff
Different interfaces used by each type of user
Casual users use an interactive query interface
The query compiler parses and validates the submitted query
The internal query is then processed for query optimization

System Catalogue & Data Dictionary:
Integrated Data Dictionary: integral part of the DBMS, fully active
Independent Data Dictionary: passive, no run-time link between data dictionary and DBMS

Data Dictionary System:

A fully functional Data Dictionary System (DDS) should store and manage:

a) Descriptions of the **database schemas**

b) Detailed information on **physical database design**
   - Storage structures
   - Access paths
   - File and record sizes

c) Descriptions of the types of **database users**, their **responsibilities** and their **access rights**

d) High-level descriptions of **transactions**, **applications** and the **relationships of users to transactions**

e) The **relationship between database transactions** and the **data items** referenced by them

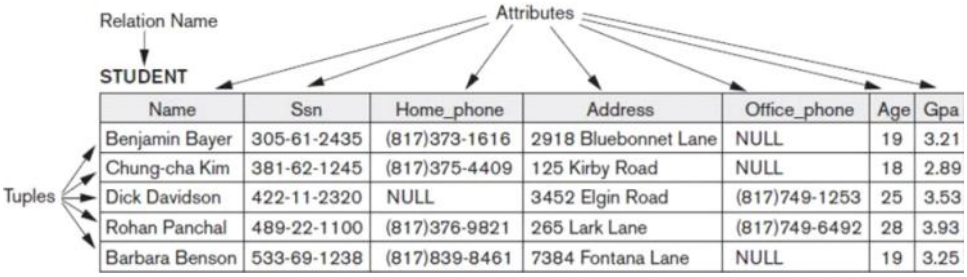
f) **Usage statistics** such as frequencies of queries and transactions and access counts to different portions of the database

g) The **history** of any changes made to the database and applications, and documentation that describes the reasons for these changes

Relational Database Model

Monday, February 17, 2025 12:09 PM

Domain: data type describing the values that can appear in each column#



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

The degree of the relation STUDENT is 7
Each value in a tuple is atomic e.g. Age
Multivalued attributes, College Degree, Must be represented by separate relations
Composite attributes, Address, Represented only by simple component attributes in basic relational model

Example: Multivalued Attributes

Person			
Person_ID	**Firstname**	**Surname**	**College Degree**
1	John	Smith	BA, PhD
2	Jane	O'Leary	BA,MSc,PhD

Person		
Firstname	**Surname**	**DOB**
1	John	Smith
2	Jane	O'Leary

College Degree		
Degree_ID	**Title**	**Person_ID**
1	BA	1
2	PhD	1
3	BA	2
4	MSc	2
5	PhD	2

Example: Composite Attribute

Person				
Person_ID	**Firstname**	**Surname**	**College Degree**	**Address**
1	John	Smith	BA, PhD	10 Doon Emma Lawns, Malahide, Co. Dublin

Person_ID	Firstname	Surname	College Degree	Street	Town	County
1	John	Smith	BA, PhD	10 Doon Emma Lawns	Malahide	Co. Dublin

NULL Values:
Values of attributes that may be unknown or may not apply to a tuple

Value exists but is not available
Value undefined

✅ Allowed NULLs:

```sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(255) NULL
);
```

- `Email` can be NULL if a customer does not provide one.

❌ Disallowed NULLs:

```sql
CREATE TABLE Salespersons (
    SalespersonID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL
);
```

- `SalespersonID` cannot be NULL because it's a primary key.
- `Name` cannot be NULL because of the **NOT NULL** constraint.

Relational Model Constraints:
Schema-based or explicit constraints:
- Key Constraints,
- Entity Integrity Constraints,
- Referential Integrity Constraints

Constraints that cannot be expressed in the schema:
- Must be enforced by the application programs
- Application-based or semantic constraints,
- Business Rules

Example: explicit constraint

1. Explicit Constraint (Enforced by the Database)

An **explicit constraint** is a rule **directly enforced** by the database schema to ensure data integrity. These constraints prevent invalid data from being entered at the database level.

Example: UNIQUE Constraint (Explicit Constraint)

```sql
```

Example: explicit constraint

1. Explicit Constraint (Enforced by the Database)

An **explicit constraint** is a rule **directly enforced** by the database schema to ensure data integrity. These constraints prevent invalid data from being entered at the database level.

Example: UNIQUE Constraint (Explicit Constraint)

```sql
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,  -- Ensures uniqueness and non-null values
    Name VARCHAR(100) NOT NULL,  -- Prevents NULL values
    Email VARCHAR(255) UNIQUE  -- Ensures no duplicate emails
);
```

- **PRIMARY KEY** ensures each `CustomerID` is unique.
- **NOT NULL** ensures the `Name` field is always filled.
- **UNIQUE** ensures no two customers have the same email.

Example: application constraint

2. Application Constraint (Enforced by the Application Code)

An **application constraint** is a rule **implemented in the application layer** (e.g., in Java, Python, or JavaScript) instead of the database. This means invalid data could still be inserted if the database is accessed directly.

Example: Business Rule Enforced in an Application (Application Constraint)

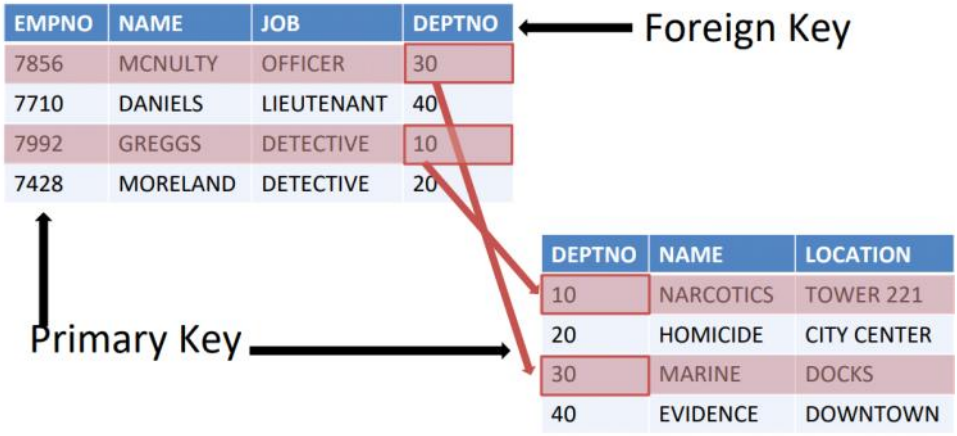
```python
def create_customer(name, email):
    if not name:
        raise ValueError("Customer name is required.")  # Enforcing NOT NULL in code

    if "@" not in email:
        raise ValueError("Invalid email format.")  # Additional validation not in DB

    query = "INSERT INTO Customers (Name, Email) VALUES (?, ?)"
    db.execute(query, (name, email))
```

- The function checks if `name` is empty before inserting (similar to `NOT NULL` in SQL).
- It ensures the email format is correct (which the database **does not enforce by default**).

Primary Key
One attribute whose values uniquely identify its tuples e.g. student number
There may not be duplicate entries in the primary key attribute
NULL values are not permitted in primary key fields
Once defined, key and entity constraints are enforced by the DBMS
Referential Integrity
Referential integrity constraints are placed on two relations
A foreign key specifies a referential integrity constraint between two relations


EMPNO
NAME
JOB
DEPTNO
7856
MCNULTY
OFFICER
30
7710
DANIELS
LIEUTENANT
40
7992
GREGGS
DETECTIVE
10
7428
MORELAND
DETECTIVE
20
Foreign Key
Primary Key
DEPTNO
NAME
LOCATION
10
NARCOTICS
TOWER 221
20
HOMICIDE
CITY CENTER
30
MARINE
DOCKS
40
EVIDENCE
DOWNTOWN

Relational Database Design

Monday, February 17, 2025 12:27 PM

Problems in Relational Models

Unnecessary Duplication: If we can delete a value in a cell, and still be able to work out What they value was, then that value was 'unnecessarily duplicated'. Unnecessary duplication creates problems when updating data as you need to ensure that all copies of such information are updated when their value(s) are changed!

Redundant data: occurs if you can delete a value without information being lost

Eliminate redundancy by table splitting

" Each attribute must have at most one value in each row"

S#	Sname	P#
S5	Wells	P1
S2	Heath	P1, P4
S7	Barron	P6
S9	Edwards	P8, P2, P6

Not possible

Normal Form Databases

1st Normal Form:

The domain of each attribute contains only atomic values and the value of each attribute contains only a single value from that domain.

Relation should have no multivalued attributes or nested relations.

Form new relations for each multivalued attribute or nested relation.

2nd Normal Form:

In addition to satisfying form 1, every non-key column is fully functionally dependent on the entire primary key.

For relations where primary key contains multiple attributes, no non key attribute should be functionally dependent on a part of the primary key.

Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.

3rd Normal Form:

In addition to satisfying form 2, no non-key attributes are transitively dependent on the entire primary key.

Relation should not have a non-key attribute functionally determined by another non key attribute (or by a set of non-key attributes). That is, there should be no transitive dependency of a non-key attribute on the primary key.

Decompose and set up a relation that includes the non-key attribute(s) that functionally determine other non-key attribute(s).

Boyce-Codd Normal Form:

All attributes in a relation should be dependent on the key, the whole key, and nothing but the key

Determinants & Identifiers

If there are rules such that duplicate values of attribute A are always associated with the same value of attribute B then attribute A is a determinant of attribute B

If each possible p# value has precisely one associated part description value (Part 4 has just one part description Nut) then we can say that p# is a determinant of part description.
Similarly, if each possible p# value has precisely one quantity in stock then we can say that p# is a determinant of part quantity in stock

Superfluous Attribute
If p# determines quantity then composite attribute {p#, p description} also determines quantity but p description is thus superfluous
We assume determinants do not contain any superfluous attributes
Transitive Determinants
If A is determinant of B & B is determinant of C then A is also determinant of C
Identifiers
No two rows in a table can have identical values throughout. Therefore an individual row can always be identified by quoting the values of all its attributes
Given a determinacy diagram, we can detect and eliminate table structures which could contain redundant data.

Transforming into Normal Form
A determinant which is not a candidate identifier is called a non-identifying determinant
To transform a badly normalised table into well normalised: create new tables such that each non identifying determinant in the old table becomes a candidate identifier in a new table

Potential Table:

Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Potential redundancy

Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Customer#	Salesman#
3	A
4	A
5	B
6	C

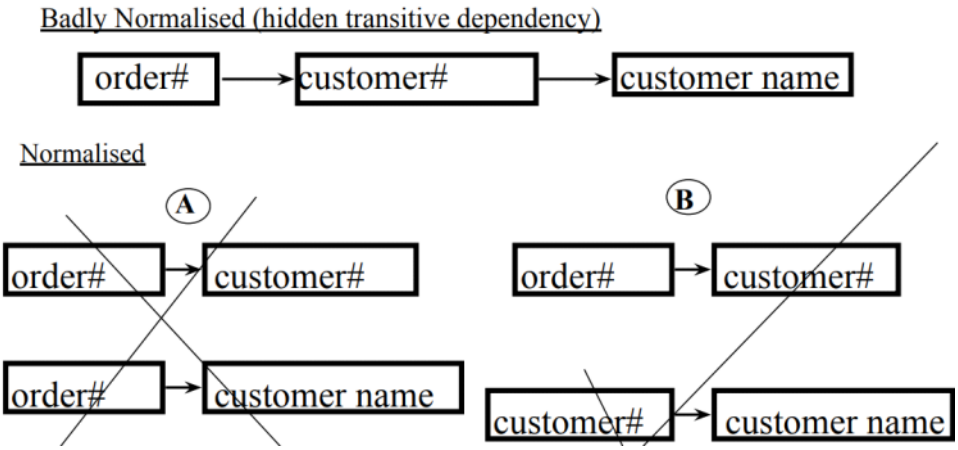
Salesman#	Salesman_name
A	John
B	Jack
C	Jim

Tables transformed into well-normalised form

Customer#	Salesman#
3	A
4	A
5	B
6	C

Salesman#	Salesman_name
A	John
B	Jack
C	Jim

Fully normalised tables are structured in such a way that they cannot contain redundant data



A

Order#	Customer#
1	c1
2	c2
3	c3

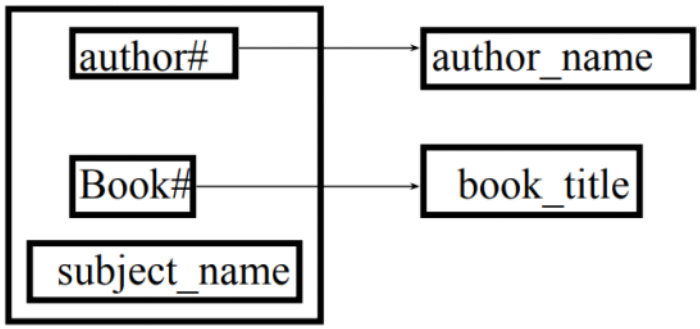
Order#	Customer_name
1	Smith
2	Jones
3	Smith

B

Order#	Customer#
1	c1
2	c2
3	c3

Customer#	Customer_name
c1	Smith
c2	Jones
c3	Smith

Multivalued Determinacy
each book has a unique book#
each author has a unique author#
every author has a name and every book has a title
each subject classification has a unique subject_name
book# does not distinguish an individual copy of a book, only an individual work
A book may be written by several authors and be classified under several subject_names
an author may write several books
a subject_name may be applied to several books



Well normalised tables:
Author (author#, author_name) ✓
Book (book#, book_title) ✓
Author_Book_Subject(author#,book#,subject_name) ✗

Car Wash Tutorial (ERD)

Wednesday, April 23, 2025 11:40 AM

Entities:

- Customer
- Vehicle
- Product
- Supplier
- Branch
- Staff
- Car wash

Relationships:

- Staff <---> Car wash – works
- Customer <---> Vehicle – owns
- Product <---> Supplier – supplies
- Branch <---> Car wash – part of
- Product <---> Staff – uses
- Vehicle <---> Staff – works on
- Customer <---> Car wash – visits

Cardinalities:

- <works> - 1 to 1
- <owns> - 1 to n
- <supplies> - 1 to n
- <part of> - 1 to 1
- <uses> - 1 to n
- <works on> - 1 to n
- <visits> - 1 to 1

Attributes:

- Customer: (name, vehicle reg (FK)) (composite PK)
- Vehicle: vehicle reg (PK), vehicle type (multi-valued attribute)
- Product: product id (PK), supplier id (FK), product name
- Supplier: supplier id (PK), phone, address
- Staff: staff ssn (PK), phone, salary
- Branch: branch id (PK), address
- Car wash: (branch id(FK), location) (composite PK), no. Of vehicles

Assumptions:

Cardinality Assumptions:

- 1 staff member can wash n number of vehicles
- 1 staff member can use n number of products
- 1 staff member can work in 1 car wash only
- 1 customer can own n vehicles
- 1 supplier can supply n products
- 1 car wash can have n customers
- 1 car wash is a part of 1 branch only

Attribute Assumptions:

- Every staff member has a social security number that serves as a primary key
- Every product supplied has a unique product id that serves as a primary key
- Every supplier has a unique supplier id that serves as a primary ley
- Every branch has a branch id that serves as a primary key
- Every car wash is uniquely identified by its branch id and location that together serve as a composite primary key
- Every customer is uniquely identified by their name and the registration of the car that is being washed which together serve as a composite primary key
- Every vehicle is uniquely identified by its registration number which serves as a primary key
- The vehicle type is a multi-valued attribute e.g. car, van, truck, motorbike.

Functional Dependencies:

- Customer: (name, vehicle reg) -> own relationship
- Vehicle: vehicle reg -> type
- Branch: branch id -> address
- Staff: staff ssn -> phone, salary

- Car wash: (branch id, location) -> no. Of vehicles
- Product: product id -> name
- Supplier: (supplier id, product id) -> supply relationship