1.Index:

a.

1. A picture containing graphical user interface

   Description automatically generated

SELECT \* FROM inventory WHERE quan\_in\_stock > 18;

b.

Graphical user interface, text, application

Description automatically generated

CREATE INDEX idx\_hash ON inventory USING HASH(quan\_in\_stock);

EXPLAIN

SELECT \* FROM inventory WHERE quan\_in\_stock > 18;

c.

Graphical user interface, text, application, email

Description automatically generated

CREATE INDEX idx\_btree ON inventory USING BTREE(quan\_in\_stock);

EXPLAIN

SELECT \* FROM inventory WHERE quan\_in\_stock > 18;

d. There is no differenece between query plans in b and c. Because numbers of products having quan\_in\_stock > 18 are very large so the DBMS considered seq scan as fastest way.

e.

Graphical user interface, text, application, email

Description automatically generated

DROP INDEX IF EXISTS idx\_hash;

DROP INDEX IF EXISTS idx\_btree;

redo b

Graphical user interface, text, application, email

Description automatically generated

redo c

Graphical user interface, text, application, email

Description automatically generated

redo d

Now, there are differences between b and c when redo because numbers of products having quan\_in\_stock > 400 are fewer.

Hash indexes only support equality comparisons when performing lookups. Bitmap index scan is useful with low cardinality and full match value.

f. There are differences between d and e when list of records in Inventory having “quan\_in\_stock” greater than 400 and 18. Because numbers of products having quan\_in\_stock > 400 are fewer than numbers of products having quan\_in\_stock > 18.

2. Query:

a.

Graphical user interface

Description automatically generated with medium confidence

SELECT DISTINCT p.prod\_id

FROM products p

JOIN inventory i

ON p.prod\_id = i.prod\_id

WHERE i.quan\_in\_stock < 100;

Graphical user interface

Description automatically generated with low confidence

SELECT DISTINCT p.prod\_id

FROM products p

WHERE p.prod\_id IN (SELECT prod\_id

FROM inventory

WHERE quan\_in\_stock < 100);

b.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

No difference between 2 ways of query. Because there is no index in quan\_in\_stock (inventory), and we have index on prod\_id of table products

c.

after create index btree on quan\_in\_stock in inventory table, it is used in both ways of query.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Because numbers of products having quan\_in\_stock < 100 are not much. Bitmap index scan is useful with low cardinality and full match value.

3. Trigger:

Graphical user interface, text, application, email

Description automatically generated

CREATE OR REPLACE FUNCTION update\_inventory()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

UPDATE inventory SET quan\_in\_stock = quan\_in\_stock - NEW.quantity, sales = sales + NEW.quantity

WHERE prod\_id = NEW.prod\_id;

ELSIF TG\_OP = 'UPDATE' THEN

UPDATE inventory SET quan\_in\_stock = quan\_in\_stock + OLD.quantity - NEW.quantity, sales = sales + NEW.quantity - OLD.quantity

WHERE prod\_id = NEW.prod\_id;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_trigger

AFTER INSERT OR UPDATE ON orderlines

FOR EACH ROW

EXECUTE FUNCTION update\_inventory();

4. Function:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with low confidence