Physical Database Design

ASSIGNMENT ONE

Peifen Lu 18008550

Christopher Li 16961502

Table of Contents

1. Business Rule 2

2. Table Definitions 2

2.1 Detailed ERD 2

2.2 Constraints Definition & Justification 3

2.3 Entity integrity and referential integrity 5

2.4 Choice & Justification of table storage params 5

2.5 Sample data for each table 8

3. Table Size 10

3.1 Calculation of Tablespace Size 10

3.2 Choice/Explanation of Tablespace Parameters and Values 12

3.3 Tablespace Creation Script 13

4. Creation Of Users, Roles, Profiles 13

4.1 Security Policy And Matrix 13

4.2 Users, Roles, Profiles – Defined in Creation Scripts, and Explained 14

4.2.1 User 14

4.2.2 Role 15

4.2.3 Profile 16

4.2.4 Users, Roles, Profiles Creation Scripts 16

5. Table Creation Scripts 20

6. Procedure to populate Customer table with Data 24

6.1 Insert Data using Insert Script 24

6.2 Insert data using Procedure: 24

6.3 View Customer Data to Confirm Procedure Works Well: 25

7. Procedure to populate Product table with Data 25

7.1 Insert Data using Insert Script: 25

7.2 Insert Data using Procedure 26

7.3 View Product Data to Confirm Procedure Works Well: 27

8. Procedure to Add a New Sale 27

9. Trigger to preload data in Despatch Table 29

10. Procedure to Update Despatch Table 30

11. Test the Procedures 30

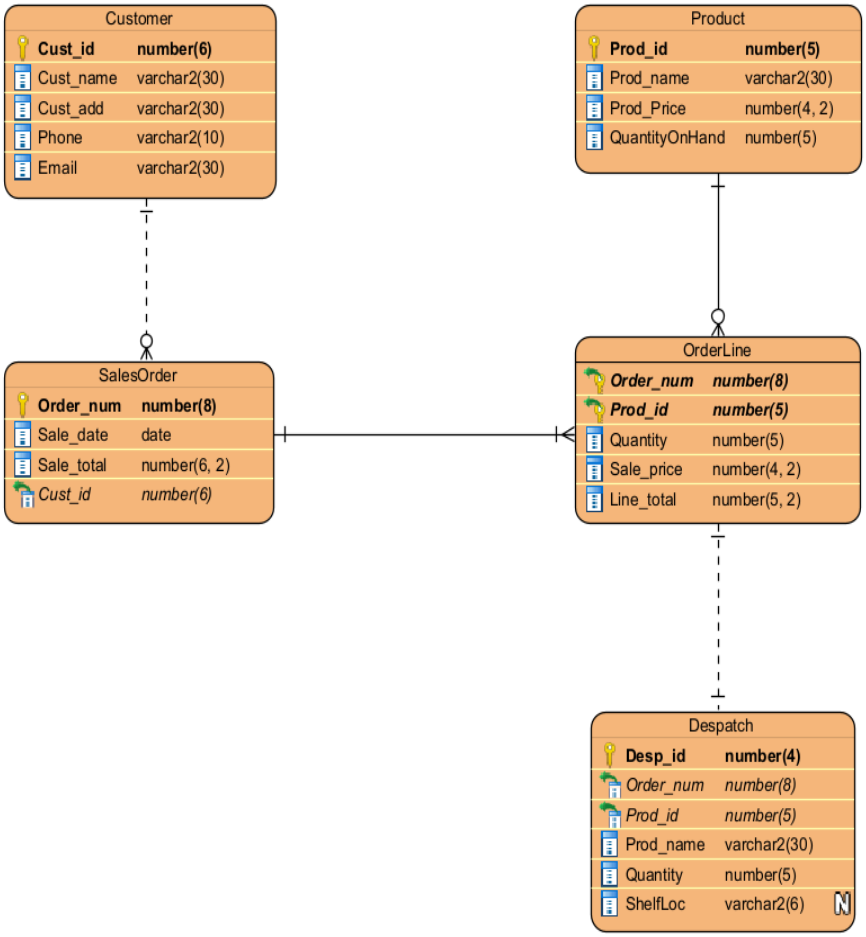
1. Business Rule

ENTITY_1	May/Must	Verb	number	ENTITY_2
Each Customer	may	Have	many	SalesOrder
Each SalesOrder	must	be placed by	only one	Customer
Each SalesOrder	must	Be related to	At least one	OrderLine
Each OrderLine	Must	Belong to	only one	SalesOrder
Each OrderLine	Must	involve	only one	Product
Each Product	May	Be involved in	many	OrderLine
Each OrderLine	Must	Be despatched by	Only one	Despatch
Each Despatch	Must	Despatch for	Only one	OrderLine

Note: OrderLine is created to solve the many to many relationships.

2. Table Definitions

2.1 Detailed ERD



2.2 Constraints Definition & Justification

Customer Table

Column name	Data Type:	Constraints	Justification
Cust_id	Number(6)	Primary Key, Not Null	Cust_id is the primary key of Customer Table. Cust_id is unique and cannot be null.
Cust_name	Varchar2(30)	Not Null	It is assumed that Best Electronics requires the customer name when the company dispatches the order to the customer. Therefore, cust_name cannot be null.
Cust_add	Varchar2(30)	Not Null	It is assumed that Best Electronics requires the customer address when the company dispatches the order to the customer. Therefore, cust_add cannot be null.
Phone	Varchar2(10)	Not Null	It is assumed that Best Electronics require the customer phone number for contact. Therefore, Phone cannot be null.
Email	Varchar2(30)	Not Null	It is assumed that Best Electronics requires the customer's email to send out invoices and dispatch details. Therefore, Email cannot be null.

SalesOrder Table

Column name	Data Type:	Constraints	Justification
Order_num	Number(8)	Primary Key, Not Null	Order_num is the primary key of the SalesOrder Table. Order_num is unique and cannot be null.
Sale_date	Date	Not Null	Best Electronics require the sale's date when processing the order. Therefore, Sale_date cannot be null.
Sale_total	Number(6,2)	Not Null	Best Electronics need the sale's total to send the invoice to the customers. Therefore, cust_add cannot be null.
Cust_id	Number(6)	Foreign key, Not Null	Cust_id is the foreign key which contains a value that refers to an existing valid tuple in Customer Table. It cannot be null because in Best Electronic, every sales order is made by a customer.

Product Table

Column name	Data Type:	Constraints	Justification
Prod_id	Number(5)	Primary Key, Not Null	Prod_id is the primary key of the Product Table. Prod_id is unique and cannot be null.
Prod_name	Number(5)	Not Null	Best Electronics need the product name for the warehouse staff to find the product. Therefore, prod_name cannot be null.
Prod_Price	Number(4, 2)	Not Null	Best Electronics need the product's price to process the order. Therefore, Prod_Price cannot be null.
QuantityOnHand	Number(5, 2)	Not Null	Best Electronics need to know the quantity remaining for the product for inventory

			management and repurchase from the suppliers. Therefore, QuantityOnHand cannot be null.

OrderLine Table

Column name	Data Type:	Constraints	Justification
Order_num	Number(8)	Composite Primary Key, Foreign Key, Not Null	Order_num is the primary key inherited from SalesOrder. Therefore Order_num is unique and cannot be null.
Prod_id	Number(5)	Composite Primary Key, Foreign Key, Not Null	Prod_id is the primary key inherited from the Product Table. Therefore Prod_id is unique and cannot be null.
Quantity	Number(5)	Not Null	Quantity is needed to process the order and therefore cannot be null.
Sale_price	Number(4,2)	Not Null	Sale's price is needed to process the order and therefore cannot be null.
Line_total	Number(5,2)	Not Null	Line_total is needed for performance improvement of queries.

Despatch Table

Column name	Data Type:	Constraints	Justification
Desp_id	Number(4)	Primary Key, Not Null	Desp_id is the primary key of the Despatch Table. Desp_id is unique and cannot be null.
Order_num	Number(8)	Foreign Key, Not Null	Order_num is the foreign key which contains a value that refers to an existing valid tuple in OrderLine Table. It cannot be null because in Best Electronic, every despatch is despatched for one orderline.
Prod_id	Number(5)	Foreign Key, Not Null	Prod_id is the foreign key which contains a value that refers to an existing valid tuple in OrderLine Table. It cannot be null because in Best Electronic, every despatch is despatched for one orderline.
Prod_name	Varchar2(30)	Not Null	As soon as the order details are saved,The product name is preloaded into the Despatch Table. Therefore Prod_name cannot be null.
Quantity	Number(5)	Not Null	As soon as the order details are saved, Quantity is preloaded into the Despatch Table. Therefore Quantity cannot be null.
ShelfLoc	Varchar2(6)		Because the shelf location for each product is added later by warehouse staff, it can be null when the despatch is created.

2.3 Entity integrity and referential integrity

TABLE	PRIMARY KEY	FOREIGN KEY(S)	ENTITY AND REFERENTIAL INTEGRITY
Customer	Cust_id		Each Cust_id is unique and there are no nulls.
SalesOrder	Order_num	Cust_id	Order_num is unique and there are no nulls.
OrderLine	Order_num, Prod_id	Order_num, Prod_id	Order_num is unique and there are no nulls. Each Order_num value in OrderLine points to an existing Order_num value in SalesOrder. Prod_id is unique and there are no nulls. Each Prod_id value in OrderLine points to an existing Prod_id value in Product.
Product	Prod_id		Each Prod_id is unique and there are no nulls.
Despatch	Desp_id	Order_num, Prod_id	Each Desp_id is unique and there are no nulls. Each Order_num value in Despatch points to an existing Order_num value in OrderLine. Each Prod_id value in Despatch points to an existing Prod_id value in OrderLine.

2.4 Choice & Justification of table storage params

- <u>Customer Table:</u>
  - Likely activities: Best Electronics have a combination of new and repeat customers and now they have 10,000 records of customers in the system. There will be new customer details inserted every day. We assume that we have 500 new customers every week. After 13 weeks, the total rows will be 16500. Customer details may be updated but are not deleted. The customers may change their address, phone or email. Therefore, data in Customer Table will need updates regularly, but not every existing customer will update their details every time they place an order. Therefore, we do not need a lot of PCTFREE.

Table storage parameters	Choice	Justification
INITIAL	3M	- Best Electronics have a combination of new and repeat customers and now they have 10,000 records of customers in the system. After a 13 weeks period, the total rows will increase to 16500. Customer details may be updated but are not deleted. - As soon as the Customer table is created, 16500 rows are loaded. Customer table will at least need 2.4 mb when first loaded. As the initial is set to be 3M, it would be good enough for the initial loading and the database does not need to extend at the initial loading process. Therefore, it is reasonable to set the initial to be 3M to maintain a good loading performance.
NEXT	1M	As the business has both repeat and new customers, customers' details will never be deleted. We set NEXT to be 1M to meet the storage needs as business grows.

PCTFREE	20%	Customer Table will need updates regularly, but not every existing customer will update their details every time they place an order. Customer details may be updated if the existing customers changes their phone, address or email or they want the order to deliver to a different address. Therefore, make PCTFREE to be 20% to reserve enough free space for updates.
PCTUSED	75%	There will be new and repeat customers in the Customer Table. As the customer details will never be deleted and there is a stable increase of customers every day, PCTUSED is set to be 75% to make sure there is enough space for inserts and existing customers.
PCTINCREASE	0	As the business has a stable amount of new customers per day and it may also have repeat customers every day. As we assume that the business will have 1000 new customers every week, we assume that NEXT is good enough for the storage needs. Therefore, we make PCTINCREASE to be zero.

- <u>SalesOrder Table:</u>
  - Likely activities: On average there are 1,000 new sales inserted per day from their customers. There will be a stable amount of new rows – 1000 per day. Sales orders are rarely updated or deleted.

Table storage params	Choice	Justification
INITIAL	4M	On average there are 1,000 new sales inserted per day from their customers and sales orders are never deleted. The total sales order table will have 91000 rows for a 13-week period, which requires 3.20 mb. We set 4MB for the initial extent to maintain a good performance when we storage the 13-week data.
NEXT	1M	It is possible that events such as public holidays, sales events that may exceed the 4M(Initial). We set NEXT to be 1M to satisfy the storage's needs.
PCTFREE	2	Because sales orders are rarely updated, we can lower PCTFREE. We set PCTFREE to be 2% instead of 0 in case of a few updates in some special circumstances.
PCTUSED	95	On average, 1000 rows are inserted per day and records of sales orders are rarely deleted. Because no rows will be deleted and 1000 rows are inserted every day, PCTUSED is set to be 95% for large amounts of storage and inserts.
PCTINCREASE	0	The business has a stable sales record of 1000 per day and it is not increasing rapidly. The need for free space is increasing stably, therefore PCTINCREASE is set to be 0.

- <u>OrderLine Table:</u>
  - Likely Activities: Because there are 1,000 sales recorded per day. On each sales order, there will be multiple orderline. We assume that when a customer places an order, he will buy two products on average.. It means that each sales order will possibly have two orderline. Therefore, there are 2,000 orderline inserted in the table on average. Because sales orders are rarely updated or deleted, orderLine are rarely updated or deleted.

Table storage parameters	Choice	Justification

INITIAL	7M	The total orderline table will have 182000 rows for one cycle (13 weeks), which requires 6.58 mb. We set 7MB for the initial extent to maintain a good performance when loading these data.
NEXT	1M	It is possible that events such as public holidays, sales events that may exceed the 7M(Initial). We set NEXT to be 1M to meet the business's needs.
PCTFREE	2%	Because sales orders are rarely updated, orderLine are rarely updated. We set PCTFREE to be 2% instead of 0 in case of a few updates in some special circumstances.
PCTUSED	95	On average there are about 3000 rows inserted per day and records of the orderline are rarely deleted. Because no rows will be deleted and 2000 rows are inserted every day, PCTUSED is set to be 95% for large amounts of storage and inserts.
PCTINCREASE	0	As the business has a stable sales record of 1000 per day, it means that the OrderLine has a stable of increase which is about 2,000 per day. The need for free space is increasing stably, therefore PCTINCREASE is set to be 0.

- <u>Product Table:</u>
  - Likely Activities: Best Electronics have a stable product list of about 100. There will be plenty of updates because for every product sold, the business needs to update the QuantityOnHand for inventory management and will need to update the Prod_Price whenever the cost of product changes. However, the Product table will barely need deletes or inserts.

Table storage parameters	Choice	Justification
INITIAL	64K	100 rows of popular products will be loaded as soon as the table is created, which requires 16K space. 64K is the default minimum extent which is good enough for Product Table.
NEXT	0	The Product is a stable table with rarely inserts. Therefore, INITIAL extent is more than enough.
PCTFREE	40%	Best Electronics will need to update the QuantityOnHand for each product sold. The update will be very often in this table. Besides, whenever the suppliers change the product price, the business will have to update the product price. Therefore, PCTFREE is set to be 40% in order to have enough free space for large amounts of updates.
PCTUSED	60%	Best Electronics have a stable product list of about 100 electronic products. The Product table may rarely need inserts or deletes. To ensure there will be enough room for current product storage, we want to fill up the block and allow the PCTUSED to be 60%.
PCTINCREASE	0	Product Table is a stable table. Deletes and inserts barely happen in this table. PCTINCREASE can be set to 0.

- <u>Despatch Table:</u>
  - Likely Activities: The Despatch table is preloaded with the product ID, product name, quantity as soon as the order details are saved. Each despatch row will be updated later with a shelf location by the warehouse staff. Therefore, there will be a large amount of updates as well as inserts every day. Because each orderline is despatched by one despatch record, as we stated earlier, each sales order will have two orderline on average. Therefore, there are 2,000 orderline inserted in the table on average. In this case, there are 2000 despatched rows inserted daily. Because sales orders and orderLine are rarely deleted, despacth records are rarely deleted.

Table storage parameters	Choice	Justification

INITIAL	17M	The total orderline table will have 182000 rows for one cycle (13 weeks), which requires 17 MB. We set INITIAL to be 17M to hold one cycle of records.
NEXT	2M	It is possible that events such as public holidays, sales events that may exceed the 21M(Initial). We set NEXT to be 2M to meet the business's needs.
PCTFREE	30%	The shelf location for each order line is updated later by the warehouse staff. Therefore, each dispatch record will need updates once. There will be 2000 rows on average that need updates every day. We should set PCTFREE to be 30% to satisfy the number of updates.
PCTUSED	70%	Because each orderline is despatched by one despatch record, as we stated earlier, each sales order will have two orderline on average. Therefore, there are 2,000 orderline inserted in the table on average. In this case, there are 2000 despatched rows inserted daily. Because sales orders and orderLine are rarely deleted, despacth records are rarely deleted. If we make PCTUSED to be 70% will be more than enough to satisfy the needs of storage and inserts.
PCTINCREASE	0	As the business has a stable sales record of 1000 per day, it means that the Despach has a stable of increase which is about 2,000 per day. The need for free space is increasing stably. There is no rapid increase of new inserted rows in this table. Thus, PCTINCREASE is set to be 0.

2.5 Sample data for each table

- Customer Table:

Cust_id	Cust_name	Cust_add	Phone	Email
1	Oliver Brown	10A Pax Avenue, Forrest Hill	023853900	obrown@hotmail.com
2	Charlie Miller	26 Rosalind Road, Glenfield	021000983	cmiller@gmail.com
3	Grace Davis	73 Moire Road, Glenfield	022994311	gdavis@hotmail.com
4	Mia Williams	9 Hutchinson Avenue, New Lynn	023999211	williams12@hotmail.com
5	Alice Jackson	191 Halsey Drive, Lynfield	021222671	alice.@gmail.com
6	Emily Ander	629A Mount Albert Road, Oak	0210895672	eander@hotmail.com

- SalesOrder Table:

Order_num	Sale_date	Sale_total	Cust_id
1	24/08/2021	50	2
2	24/08/2021	50	6
3	24/08/2021	30	3
4	24/08/2021	22	4
5	24/08/2021	39	1

- OrderLine Table:

Order_num	Prod_id	Quantity	Sale_price	Line_total
1	1	1	20	20
1	2	1	30	30
2	2	1	30	30
2	3	2	10	20
3	3	1	10	10
3	4	2	10	20
4	4	1	10	10
4	5	1	12	12
5	5	2	12	24
5	6	1	15	15

- Product Table:

Prod_id	Prod_name	Prod_Price	QuantityOnHand
1	PowerPlay Gaming Mouse	20	20
2	Logitech Keyboard	30	50
3	PowerPlay Mouse Pad	10	30
4	XCD earphones	10	60
5	SADES earphones	12	30
6	PowerPlay Memory Card	15	59

- Despatch Table:

Desp_id	Order_num	Prod_id	Prod_name	Quantity	ShelfLoc
1	**1**	1	PowerPlay Gaming Mouse	1	S-A
2	**1**	2	Logitech Keyboard	1	S-B
3	**2**	2	Logitech Keyboard	1	S-B
4	**2**	3	PowerPlay Mouse Pad	2	S-C
5	**3**	3	PowerPlay Mouse Pad	1	S-C
6	**3**	4	XCD earphones	2	S-D
7	**4**	4	XCD earphones	1	S-D
8	**4**	5	SADES earphones	1	S-E
9	**5**	5	SADES earphones	2	S-E
10	**5**	6	PowerPlay Memory Card	1	S-F

3. Table Size

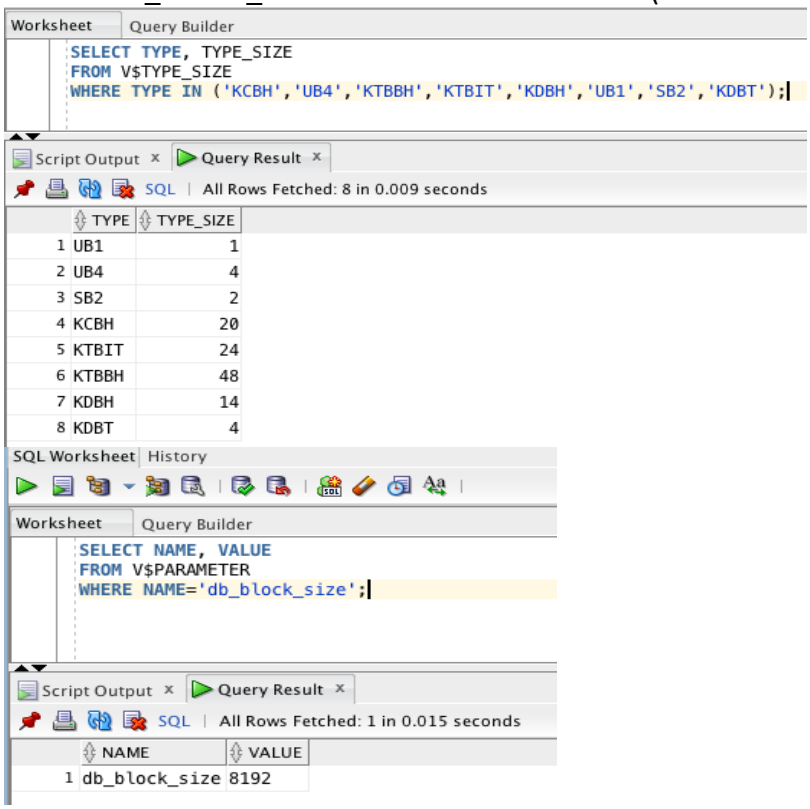
3.1 Calculation of Tablespace Size

To calculate Space Required:

1. Calculate the total block header size

   To calculate total space after header size (HSIZE), the formula:

   HSIZE = DB_BLOCK_SIZE – KCBH – UB4 – KTBBH – (INITRANS-1)*KTBIT – KDBH



   DB_BLOCK_SIZE:8192
   KCBH: 20
   UB4: 4
   KTBBH: 48
   KTBIT: 24
   KDBH: 14
   INITRANS: 1 (default value)

   Therefore, HSIZE = 8192-20-4-48-(1-1)*24-14=8106

2. Calculate the available data space per data block

From the script early, we know that KDBT = 4

Available Data Space = CEIL(hsize*(1-PCTFREE/100)) – KDBT

Table	PCTFREE	Available Data Space
Customer	20	=CEIL(8106*(1-20/100))-4=6481 bytes
SalesOrder	2	=CEIL(8106*(1-2/100))-4=7940 bytes
OrderLine	2	=CEIL(8106*(1-2/100))-4=7940 bytes
Product	40	=CEIL(8106*(1-40/100))-4=4860 bytes
Despatch	30	=CEIL(8106*(1-30/100))-4=5671 bytes

3. Calculate the space used per row

Table	Column Sizes	Column Size + Byte Length	(Total) Column Size
Customer	=6+30+30+10+30=106	=(6+1)+(30+1)+(30+1)+(10+1)+(30+1)=111	111
SalesOrder	=8+7+6+6=27	=(8+1)+(7+1)+(6+1)+(6+1)=31	31
OrderLine	=8+5+5+4+5=27	=(8+1)+(5+1)+(5+1)+(4+1)+(5+1)=32	32
Product	=5+30+4+5=44	=(5+1)+(30+1)+(4+1)+(5+1)=48	48
Despatch	=4+8+5+30+5+6=58	=(4+1)+(8+1)+(5+1)+(30+1)+(5+1)+(6+1)=64	64

Table	row header = 3*UB1	Column Size(Total)	rowsize = row header + sum of all column sizes	rowspace = max(row header+UB4+SB2,rowsize)+SB2
Customer	=3*1=3	111	=3+111=114	=max(3+4+2,114)+2=116
SalesOrder	=3*1=3	31	=3+31=34	=max(3+4+2,34)+2=36
OrderLine	=3*1=3	32	=3+32=35	=max(3+4+2,35)+2=37
Product	=3*1=3	48	=3+48=51	=max(3+4+2,51)+2=53
Despatch	=3*1=3	64	=3+64=67	=max(3+4+2,67)+2=69

4. Calculate the total number of rows that will fit in a data block

Table	ROW COUNT per cycle	Number of rows in block (RB) = FLOOR(availspace/rowspace)	Total number of blocks required: CEIL(total rows needed/number of rows in block)
Customer	=10000+500*13= 16500	=FLOOR(6481/116)=55	=CEIL(16500/55)=300
SalesOrder	=1000*7*13=91000	=FLOOR(7940/36)=220	=CEIL(91000/220)=414
OrderLine	=2000*7*13= 182000	=FLOOR(7940/37)=214	=CEIL(182000/214)=851
Product	100	=FLOOR(4860/53)=91	=CEIL(100/91)=2
Despatch	=2000*7*13= 182000	=FLOOR(5671/69)=82	=CEIL(182000/82)=2220

5. Calculate the tablespace size

Table	Total number of blocks	DB BLOCK SIZE(8106 KB)	Table Size
Customer	300	8106 bytes	=8106*300=2431800 bytes=2375kb
SalesOrder	414	8106 bytes	=8106*414=3355884 bytes=3277kb
OrderLine	851	8106 bytes	=8106*851=6898206 bytes=6737kb
Product	2	8106 bytes	=8106*2=16212 bytes=16kb
Despatch	2220	8106 bytes	=8106*2220=17995320 bytes=17574kb

3.2 Choice/Explanation of Tablespace Parameters and Values

Table	Tablespace size	Justification
Customer	2375kb	Because the customer base is relatively stable. The business has a combination of new and repeat customers. We assume that we have 500 new customers every week. There will be new customer details inserted every day. In this case, an auto extensible file is needed as the insertion of new data is frequent.
SalesOrder	3277kb	On average there are 1,000 new sales inserted per day from their customers and sales orders are never deleted. As there may be frequent new data insertion, we need an auto extensible file for the SalesOrder table.
OrderLine	6737kb	On average there are 1,000 new sales inserted per day from their customers and sales orders are never deleted. As we assume that each sales order contains two orderline records on average, it means that there are 2,000 new orderline data inserted on average. There will be frequent new data insertion, so we need an auto extensible file for the OrderLine table.
Product	16kb	Best Electronics have a stable product list of about 100 electronic products. The Product table may rarely need inserts or deletes. In this situation, an auto extensible file is not needed.
Despatch	17574 kb	There are new rows inserted every day, an auto extensible file is needed.
Total Tablespace	29978 Kb	

TOTAL TABLESPACE SIZE = 2375KB+3277KB+6737KB+16KB+17574KB=29978KB=30MB

The total tablespace needed for these five table is 29978 KB which is about 30 MB. Instead of creating five tablespaces for each table, we will create 1 tablespaces for 5 tables. Because Customer, SalesOrder, OrderLine, Product and Despatch are all related, we will have 1 tablespace for these five tables. And we set the autoextend to be on for this tablespace because there will be frequent data inserted in OrderLine, SalesOrder and Despatch Table every day.

3.3 Tablespace Creation Script

Worksheet | Query Builder

```
CREATE TABLESPACE gkn3798_ts_32
DATAFILE '/usr/local/apps/oracle/oradata/ldb30/gkn3798_ts32.dat'
SIZE 30M
AUTOEXTEND ON;
```

Script Output | Task completed in 1.265 seconds

```
TABLESPACE GKN3798_TS_32 created.
```

4. Creation Of Users, Roles, Profiles

4.1 Security Policy And Matrix

Authorisation for DBA

	Customer Table	SalesOrder Table	OrderLine Table	Product Table	Despatch Table
Read	√	√	√	√	√
Insert	√	√	√	√	√
Modify	√	√	√	√	√
Delete	√	√	√	√	√
Justification	In order create and maintain the database, DBA need full access to the database.				

Authorisation for Developers

	Customer Table	SalesOrder Table	OrderLine Table	Product Table	Despatch Table
Read	√	√	√	√	√
Insert	√	√	√	√	√
Modify	√	√	√	√	√
Delete	√	√	√	√	√
Justification	In order for the developers to develop a front-end and back-end for the database, the developers will need to have object privileges-read, insert, modify and delete on Customer, SalesOrder, OrderLine, Product and Despatch tables.				

System Privileges

User Type	System Privileges	Justification
DBA	All system privileges with admin option	DBA are expected to perform the tasks including creating the database, backup the database, enrolling system users, implementing database design and tuning database performance, so DBA needs all system privileges.
Developers	Create Table Alter Table Drop Table Create any Table Alter any Table Drop any Table Insert any Table Delete any Table Update any Table Select any Table	-When the developers are doing the development, they will create the tables with the same schemas to work and test with. Therefore, they need to have the create, alter, drop, insert, delete, update, select (any) table privilege.
	Create any Index Alter any Index Drop any index	-Developers need create, alter, drop any index to do the following development tasks.
	Create View Drop View	-The developers need create, drop view for development.
	Create Session Alter Session	-The developers need create, alter session to connect to the database.
	Create Procedure Alter Procedure Drop Procedure Create any Procedure Alter any Procedure Execute any Procedure	-The developers need create, alter, drop procedures, create, alter and execute any procedure for development tasks.
	Create Function Alter Function Drop Function	-The developers need create, alter, drop function for development tasks.
	Create any Trigger Alter any Trigger Drop any Trigger	-The developers need create, alter, drop trigger for development tasks.
	Create any Sequence Alter any Sequence Drop any Sequence Select any Sequence	-The developers need create, alter, drop, select any sequence for development.

4.2 Users, Roles, Profiles – Defined in Creation Scripts, and Explained

4.2.1 User

Explanation

User Type	Parameters	Justification
Developer	username: gkn3798_user	The user login with a username and cannot be more than 30 bytes. It has to start with a letter and is not able to contain any special characters.
	Password: PeifenAndChris	The user logs in with a password as the authentication method.
	Default tablespace:	The user needs a default tablespace. Whenever the user creates a table or upload the data, the data will be stored here.
	Quota on tablespace: unlimited	In order for the developer to implement the development and upload the data, we assign full quota on tablespaces.
	Temporary tablespace: temp	temp as default temporary tablespace to store all the temporary data created by the developer.
	Role:gkn3798_dev	We will assign privilege and a developer role to the user.
	Profile:gkn3798_profile	We will assign a profile to the user to control the access rights of this user and make some password restrictions as well. Each user can have only one profile.
	Account status: unlock	We set the account status to be unlock so that this user account can be unlocked and used by the user.

User Type	Parameters	Justification
DBA	username: gkn3798_dbauser	The DBA login with a username and cannot be more than 30 bytes. It has to start with a letter and is not able to contain any special characters.
	Password: peifendba	The DBA logs in with a password as the authentication method.
	Default tablespace:	The DBA needs a default tablespace. Whenever the DBA creates a table or upload the data, the data will be stored here.
	Quota on tablespace: unlimited	In order for the dba to maintain the database and upload the data, we assign full quota on tablespaces.
	Temporary tablespace: temp	temp as default temporary tablespace to store all the temporary data created by the DBA.
	Role:gkn3798_dba	We will assign privilege and a developer role to the user.
	Profile:gkn3798_profile	We will assign the same profile to the dba user to control the access rights of this user and make some password restrictions as well. Each user can have only one profile.
	Account status: unlock	We set the account status to be unlock so that this user account can be unlocked and used by the user.

4.2.2 Role

Explanation

Because there will be different types of users in this organisation, we set up roles with a group of privileges to manage the privileges easier. We create a dba and developer role with a group of privileges we defined earlier to these two roles and grant the developer role to the developer user, grant the dba role to the dba user.

4.2.3 Profile

Explanation

Parameters	Value	Justification
SESSIONS_PER_USER	2	We set SESSIONS_PER_USER to be 2. The user can only open two sessions concurrently.
CPU_PER_SESSION	unlimited	CPU time limit for a session. We set it to be unlimited as it is for a developer user.
LOGICAL_READS_PER_SESSION	unlimited	The permitted number of data blocks read in a session. We set it to be unlimited as it is for developer user.
IDLE_TIME	20	The user can only stay idle for 20 mins. If more than 20 mins, oracle database will close the session.
CONNECT_TIME	480	The total elapsed time limit for a session is 480 minutes. The user will only allow to connect to the session during his working hours which is 8 hours.
FAILED_LOGIN_ATTEMPTS	5	To We make the profile to be locked after 5 attempts
PASSWORD_LOCK_TIME	3	Once the user fails to login for 5 times, the account will be locked for 3 days. And then he can try to login the account.
PASSWORD_LIFE_TIME	180	To ensure security, we set PASSWORD_LIFE_TIME to be 180 to make the user change the password every 180 days.
PASSWORD_GRACE_TIME	7	We set PASSWORD_GRACE_TIME to be 7 to give the user 7 days to change the password.
PASSWORD_REUSE_TIME	UNLIMITED	We set PASSWORD_REUSE_TIME to be unlimited to ensure the safety. The user can never use the same password.
PASSWORD_REUSE_MAX	UNLIMITED	The number of password changes that are required before the current passwords can be reused. We set it to be unlimited. The user can never use the same password.

We assign the same profile to dba and developer.

4.2.4 Users, Roles, Profiles Creation Scripts

<u>Create Profile Scripts:</u>

```
CREATE PROFILE gkn3798_profile LIMIT
    SESSIONS_PER_USER 2
    CPU_PER_SESSION unlimited
    LOGICAL_READS_PER_SESSION unlimited
    IDLE_TIME 20
    CONNECT_TIME 480
    FAILED_LOGIN_ATTEMPTS 5
    PASSWORD_LOCK_TIME 3
    PASSWORD_LIFE_TIME 180
    PASSWORD_GRACE_TIME 7
    PASSWORD_REUSE_TIME UNLIMITED
    PASSWORD_REUSE_MAX UNLIMITED;
```

Script Output

Task completed in 0.028 seconds

```
Profile GKN3798_PROFILE created.
```

Create User (for developer), Assign Profile to User (developer) Scripts:

Worksheet | Query Builder

```
CREATE USER gkn3798_user
IDENTIFIED BY PeifenAndChris
DEFAULT TABLESPACE gkn3798_ts_32
TEMPORARY TABLESPACE temp
ACCOUNT UNLOCK
PROFILE gkn3798_profile;
alter user gkn3798_user quota unlimited on gkn3798_ts_32;
```

Script Output

Task completed in 0.029 seconds

```
User GKN3798_USER created.

User GKN3798_USER altered.
```

Create Developer Role, Grant Privilege to Developer Role, Grant Developer Role to Developer User Scripts:

```
CREATE ROLE gkn3798_dev;
GRANT CREATE TABLE TO gkn3798_dev;
GRANT CREATE ANY TABLE TO gkn3798_dev;
GRANT ALTER ANY TABLE TO gkn3798_dev;
GRANT DROP ANY TABLE TO gkn3798_dev;
GRANT INSERT ANY TABLE TO gkn3798_dev;
GRANT DELETE ANY TABLE TO gkn3798_dev;
GRANT UPDATE ANY TABLE TO gkn3798_dev;
GRANT SELECT ANY TABLE TO gkn3798_dev;
GRANT CREATE ANY INDEX TO gkn3798_dev;
GRANT ALTER ANY INDEX TO gkn3798_dev;
GRANT DROP ANY INDEX TO gkn3798_dev;
GRANT CREATE VIEW TO gkn3798_dev;
GRANT CREATE ANY VIEW TO gkn3798_dev;
GRANT DROP ANY VIEW TO gkn3798_dev;
GRANT CREATE SESSION TO gkn3798_dev;
GRANT ALTER SESSION TO gkn3798_dev;
GRANT CREATE PROCEDURE TO gkn3798_dev;
GRANT DROP ANY PROCEDURE TO gkn3798_dev;
GRANT CREATE ANY PROCEDURE TO gkn3798_dev;
GRANT ALTER ANY PROCEDURE TO gkn3798_dev;
GRANT EXECUTE ANY PROCEDURE TO gkn3798_dev;
GRANT CREATE ANY TRIGGER TO gkn3798_dev;
GRANT ALTER ANY TRIGGER TO gkn3798_dev;
GRANT DROP ANY TRIGGER TO gkn3798_dev;
GRANT CREATE ANY SEQUENCE TO gkn3798_dev;
GRANT ALTER ANY SEQUENCE TO gkn3798_dev;
GRANT DROP ANY SEQUENCE TO gkn3798_dev;
GRANT SELECT ANY SEQUENCE TO gkn3798_dev;
```

Script Output | Task completed in 0.022 seconds

```
Grant succeeded.
```

Worksheet | Query Builder

```
GRANT gkn3798_dev TO gkn3798_user;
```

Script Output | Query Result | Task completed in 0.024 seconds

```
Grant succeeded.
```

Create User (for DBA), Assign Profile to User (DBA) Scripts:

Worksheet | Query Builder

```
CREATE USER gkn3798_dbauser
IDENTIFIED BY peifendba
DEFAULT TABLESPACE gkn3798_ts_32
TEMPORARY TABLESPACE temp
ACCOUNT UNLOCK
PROFILE gkn3798_profile;
alter user gkn3798_dbauser quota unlimited on gkn3798_ts_32;
```

Script Output | Task completed in 0.031 seconds

```
User GKN3798_DBAUSER created.
```

Script Output

Task completed in 0.031 seconds

```
User GKN3798_DBAUSER altered.
```

Create DBA Role, Grant Privilege to DBA Role:

Worksheet | Query Builder

```
CREATE ROLE gkn3798_dba;
GRANT CREATE TABLESPACE TO gkn3798_dba;
GRANT ALTER TABLESPACE TO gkn3798_dba;
GRANT DROP TABLESPACE TO gkn3798_dba;
GRANT MANAGE TABLESPACE TO gkn3798_dba;
GRANT CREATE PROFILE TO gkn3798_dba;
GRANT ALTER PROFILE TO gkn3798_dba;
GRANT DROP PROFILE TO gkn3798_dba;
GRANT CREATE ROLE TO gkn3798_dba;
GRANT ALTER ANY ROLE TO gkn3798_dba;
GRANT DROP ANY ROLE TO gkn3798_dba;
GRANT GRANT ANY ROLE TO gkn3798_dba;
GRANT CREATE USER TO gkn3798_dba;
GRANT ALTER USER TO gkn3798_dba;
GRANT DROP USER TO gkn3798_dba;
GRANT CREATE TABLE TO gkn3798_dba;
GRANT CREATE ANY TABLE TO gkn3798_dba;
GRANT ALTER ANY TABLE TO gkn3798_dba;
GRANT DROP ANY TABLE TO gkn3798_dba;
GRANT INSERT ANY TABLE TO gkn3798_dba;
GRANT DELETE ANY TABLE TO gkn3798_dba;
GRANT UPDATE ANY TABLE TO gkn3798_dba;
GRANT SELECT ANY TABLE TO gkn3798_dba;
GRANT CREATE ANY INDEX TO gkn3798_dba;
GRANT ALTER ANY INDEX TO gkn3798_dba;
GRANT DROP ANY INDEX TO gkn3798_dba;
GRANT CREATE VIEW TO gkn3798_dba;
GRANT CREATE ANY VIEW TO gkn3798_dba;
GRANT DROP ANY VIEW TO gkn3798_dba;
GRANT CREATE SESSION TO gkn3798_dba;
GRANT ALTER SESSION TO gkn3798_dba;
GRANT CREATE PROCEDURE TO gkn3798_dba;
GRANT DROP ANY PROCEDURE TO gkn3798_dba;
GRANT CREATE ANY PROCEDURE TO gkn3798_dba;
GRANT ALTER ANY PROCEDURE TO gkn3798_dba;
GRANT EXECUTE ANY PROCEDURE TO gkn3798_dba;
GRANT CREATE ANY TRIGGER TO gkn3798_dba;
GRANT ALTER ANY TRIGGER TO gkn3798_dba;
GRANT DROP ANY TRIGGER TO gkn3798_dba;
GRANT CREATE ANY SEQUENCE TO gkn3798_dba;
GRANT ALTER ANY SEQUENCE TO gkn3798_dba;
GRANT DROP ANY SEQUENCE TO gkn3798_dba;
GRANT SELECT ANY SEQUENCE TO gkn3798_dba;
```

Script Output

Task completed in 0.026 seconds

```
Role GKN3798_DBA created.
```

Script Output

Task completed in 0.026 seco

```
Grant succeeded.
```

Grant DBA Role to DBA User Scripts:

```
Worksheet   Query Builder
GRANT gkn3798_dba TO gkn3798_dbauser;

Script Output
Task completed in 0.038 seconds

Grant succeeded.
```

5. Table Creation Scripts

Customer Table:

```
CREATE TABLE Customer (
    Cust_id number(6) NOT NULL,
    Cust_name varchar2(30) NOT NULL,
    Cust_add varchar2(30) NOT NULL,
    Phone varchar2(10) NOT NULL,
    Email varchar2(30) NOT NULL,
    CONSTRAINT Customer_PK PRIMARY KEY (Cust_id)
    )
PCTFREE 20
PCTUSED 75
STORAGE (
        INITIAL 3M
        NEXT 1M
        PCTINCREASE 0
        )
TABLESPACE gkn3798_ts_32;

Script Output
Task completed in 0.052 seconds

Table CUSTOMER created.
```

SalesOrder Table:

```
CREATE TABLE SalesOrder (
    Order_num number(8) NOT NULL,
    Sale_date date NOT NULL,
    Sale_total number(6,2) NOT NULL,
    Cust_id number(6) NOT NULL,
    CONSTRAINT SalesOrder_PK PRIMARY KEY (Order_num),
    CONSTRAINT SalesOrder_Cust_id_FK FOREIGN KEY (Cust_id) REFERENCES Customer (Cust_id)
    )
PCTFREE 2
PCTUSED 95
STORAGE (
        INITIAL 4M
        NEXT 1M
        PCTINCREASE 0
        )
TABLESPACE gkn3798_ts_32;

Script Output
Task completed in 0.042 seconds

Table SALESORDER created.
```

Product Table:

Worksheet | Query Builder

```
CREATE TABLE Product (
    Prod_id number(5) NOT NULL,
    Prod_name varchar2(30) NOT NULL,
    Prod_price number(4,2) NOT NULL,
    QuantityOnHand number(5) NOT NULL,
    CONSTRAINT Product_PK PRIMARY KEY (Prod_id)
    )
PCTFREE 40
PCTUSED 60
STORAGE (
        INITIAL 64K
        NEXT 0K
        PCTINCREASE 0
        )
TABLESPACE gkn3798_ts_32;
```

Script Output | Task completed in 0.049 seconds

```
Table PRODUCT created.
```

OrderLine Table:

```
CREATE TABLE OrderLine (
    Order_num number(8),
    Prod_id number(5),
    Quantity number(5) NOT NULL,
    Sale_price number(4,2) NOT NULL,
    Line_total number(5,2) NOT NULL,
    CONSTRAINT OrderLine_pk PRIMARY KEY (Order_num, Prod_id),
    CONSTRAINT OrderLine_Order_num_FK FOREIGN KEY (Order_num) REFERENCES SalesOrder (Order_num),
    CONSTRAINT OrderLine_Prod_id_FK FOREIGN KEY (Prod_id) REFERENCES Product (Prod_id)
    )
PCTFREE 2
PCTUSED 95
STORAGE (
        INITIAL 7M
        NEXT 1M
        PCTINCREASE 0
        )
TABLESPACE gkn3798_ts_32;
```

Script Output | Task completed in 0.05 seconds

```
Table ORDERLINE created.
```

Despatch Table:

```
CREATE TABLE Despatch (
    Desp_id number(4) NOT NULL,
    Order_num number(8) NOT NULL,
    Prod_id number(5) NOT NULL,
    Prod_name varchar2(30) NOT NULL,
    Quantity number(5) NOT NULL,
    ShelfLoc varchar2(6),
    CONSTRAINT Despatch_pk PRIMARY KEY (Desp_id)
    )
PCTFREE 30
PCTUSED 70
STORAGE (
        INITIAL 17M
        NEXT 2M
        PCTINCREASE 0
        )
TABLESPACE gkn3798_ts_32;
ALTER TABLE Despatch ADD CONSTRAINT Despatch_OrderLine_fk FOREIGN KEY (Order_num, Prod_id) REFERENCES OrderLine (Order_num, Prod_id);
```

Script Output | Task completed in 0.028 seconds

```
Table DESPATCH created.

Table DESPATCH altered.
```

Verifying Constraints for each Table:

Customer Table:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'CUSTOMER';
```

Script Output | Query Result | SQL | All Rows Fetched: 6 in 0.005 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	SYS_C00500756	C	"CUST_ID" IS NOT NULL
2	SYS_C00500757	C	"CUST_NAME" IS NOT NULL
3	SYS_C00500758	C	"CUST_ADD" IS NOT NULL
4	SYS_C00500759	C	"PHONE" IS NOT NULL
5	SYS_C00500760	C	"EMAIL" IS NOT NULL
6	CUSTOMER_PK	P	(null)

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'CUSTOMER';
```

Script Output | Query Result | SQL | All Rows Fetched: 6 in 0.005 seconds

	CONSTRAINT_NAME	COLUMN_NAME
1	CUSTOMER_PK	CUST_ID
2	SYS_C00500756	CUST_ID
3	SYS_C00500757	CUST_NAME
4	SYS_C00500758	CUST_ADD
5	SYS_C00500759	PHONE
6	SYS_C00500760	EMAIL

SalesOrder Table:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'SALESORDER';
```

Script Output | Query Result | SQL | All Rows Fetched: 6 in 0.012 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	SYS_C00500762	C	"ORDER_NUM" IS NOT NULL
2	SYS_C00500763	C	"SALE_DATE" IS NOT NULL
3	SYS_C00500764	C	"SALE_TOTAL" IS NOT NULL
4	SYS_C00500765	C	"CUST_ID" IS NOT NULL
5	SALESORDER_PK	P	(null)
6	SALESORDER_CUST_ID_FK	R	(null)

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'SALESORDER';
```

Script Output | Query Result | SQL | All Rows Fetched: 6 in 0.021 seconds

	CONSTRAINT_NAME	COLUMN_NAME
1	SALESORDER_CUST_ID_FK	CUST_ID
2	SALESORDER_PK	ORDER_NUM
3	SYS_C00500762	ORDER_NUM
4	SYS_C00500763	SALE_DATE
5	SYS_C00500764	SALE_TOTAL
6	SYS_C00500765	CUST_ID

OrderLine Table:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'ORDERLINE';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 6 in 0.011 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	SYS_C00500774	C	"SALE_PRICE" IS NOT NULL
2	SYS_C00500775	C	"LINE_TOTAL" IS NOT NULL
3	ORDERLINE_PK	P	(null)
4	ORDERLINE_ORDER_NUM_FK	R	(null)
5	ORDERLINE_PROD_ID_FK	R	(null)
6	SYS_C00500773	C	"QUANTITY" IS NOT NULL

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'ORDERLINE';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 7 in 0.015 seconds

	CONSTRAINT_NAME	COLUMN_NAME
1	ORDERLINE_ORDER_NUM_FK	ORDER_NUM
2	ORDERLINE_PK	ORDER_NUM
3	ORDERLINE_PK	PROD_ID
4	ORDERLINE_PROD_ID_FK	PROD_ID
5	SYS_C00500773	QUANTITY
6	SYS_C00500774	SALE_PRICE
7	SYS_C00500775	LINE_TOTAL

Product Table:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'PRODUCT';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 5 in 0.012 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	SYS_C00500768	C	"PROD_ID" IS NOT NULL
2	SYS_C00500769	C	"PROD_NAME" IS NOT NULL
3	SYS_C00500770	C	"PROD_PRICE" IS NOT NULL
4	SYS_C00500771	C	"QUANTITYONHAND" IS NOT NULL
5	PRODUCT_PK	P	(null)

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'PRODUCT';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 5 in 0.01 seconds

	CONSTRAINT_NAME	COLUMN_NAME
1	PRODUCT_PK	PROD_ID
2	SYS_C00500768	PROD_ID
3	SYS_C00500769	PROD_NAME
4	SYS_C00500770	PROD_PRICE
5	SYS_C00500771	QUANTITYONHAND

Despatch Table:

```
SELECT constraint_name, constraint_type, search_condition
FROM user_constraints
WHERE table_name = 'DESPATCH';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 7 in 0.008 seconds

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	SYS_C00500779	C	"DESP_ID" IS NOT NULL
2	SYS_C00500780	C	"ORDER_NUM" IS NOT NULL
3	SYS_C00500781	C	"PROD_ID" IS NOT NULL
4	SYS_C00500782	C	"PROD_NAME" IS NOT NULL
5	SYS_C00500783	C	"QUANTITY" IS NOT NULL
6	DESPATCH_PK	P	(null)
7	DESPATCH_ORDERLINE_FK	R	(null)

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'DESPATCH';
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 8 in 0.013 seconds

	CONSTRAINT_NAME	COLUMN_NAME
1	DESPATCH_ORDERLINE_FK	ORDER_NUM
2	DESPATCH_ORDERLINE_FK	PROD_ID
3	DESPATCH_PK	DESP_ID
4	SYS_C00500779	DESP_ID
5	SYS_C00500780	ORDER_NUM
6	SYS_C00500781	PROD_ID
7	SYS_C00500782	PROD_NAME
8	SYS_C00500783	QUANTITY

6. Procedure to populate Customer table with Data

6.1 Insert Data using Insert Script

```
INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (1, 'Oliver Brown', '10A Pax Avenue,Forrest Hill', '023853900', 'obrown@hotmail.com');

INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (2, 'Charles Miller', '26 Rosalind Road,Glenfield', '021000983', 'cmiller@gmail.com');

INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (3, 'Grace Davis', '73 Moire Road,Glenfield', '022994311', 'gdavis@hotmail.com');

INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (4, 'Mia Williams', '9 Hutchinson Avenue,New Lynn', '023999211', 'williams12@hotmail.com');

INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (5, 'Alice Jackson', '191 Halsey Drive,Lynfield', '021222671', 'alice@gmail.com');

INSERT INTO Customer (Cust_id, Cust_name, Cust_add, Phone, Email)
    VALUES (6, 'Emily Ander', '629A Mount Albert Road,Oak', '0210895672', 'eander@hotmail.com');

select * from customer;
```

Script Output | Query Result

SQL | All Rows Fetched: 6 in 0.006 seconds

	CUST_ID	CUST_NAME	CUST_ADD	PHONE	EMAIL
1	1	Oliver Brown	10A Pax Avenue,Forrest Hill	023853900	obrown@hotmail.com
2	2	Charles Miller	26 Rosalind Road,Glenfield	021000983	cmiller@gmail.com
3	3	Grace Davis	73 Moire Road,Glenfield	022994311	gdavis@hotmail.com
4	4	Mia Williams	9 Hutchinson Avenue,New Lynn	023999211	williams12@hotmail.com
5	5	Alice Jackson	191 Halsey Drive,Lynfield	021222671	alice@gmail.com
6	6	Emily Ander	629A Mount Albert Road,Oak	0210895672	eander@hotmail.com

6.2 Insert data using Procedure:

```
create or replace PROCEDURE addNewCust(
    v_custid IN customer.cust_id%TYPE,
    v_name IN customer.cust_name%TYPE,
    v_custadd IN customer.cust_add%TYPE,
    v_phone IN customer.phone%TYPE,
    v_email IN customer.email%TYPE
) IS
BEGIN
    INSERT INTO customer
        VALUES (v_custid, v_name, v_custadd, v_phone, v_email);
    dbms_output.put_line('insert successfully');
END addNewCust;

BEGIN
  addNewCust(7,'Andy Li','7 Wall St,Henderson','02298732','ali@gmail.com');
END;
```

Script Output

Task completed in 0.041 seconds

```
Procedure ADDNEWCUST compiled


PL/SQL procedure successfully completed.
```

6.3 View Customer Data to Confirm Procedure Works Well:

	CUST_ID	CUST_NAME	CUST_ADD	PHONE	EMAIL
1	1	Oliver Brown	10A Pax Avenue,Forrest Hill	023853900	obrown@hotmail.com
2	2	Charles Miller	26 Rosalind Road,Glenfield	021000983	cmiller@gmail.com
3	3	Grace Davis	73 Moire Road,Glenfield	022994311	gdavis@hotmail.com
4	4	Mia Williams	9 Hutchinson Avenue,New Lynn	023999211	williams12@hotmail.com
5	5	Alice Jackson	191 Halsey Drive,Lynfield	021222671	alice@gmail.com
6	6	Emily Ander	629A Mount Albert Road,Oak	0210895672	eander@hotmail.com
7	7	Andy Li	7 Wall St,Henderson	02298732	ali@gmail.com

7. Procedure to populate Product table with Data

7.1 Insert Data using Insert Script:

```sql
INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (1, 'PowerPlay Gaming Mouse', 20, 20);

INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (2, 'Logitech Keyboard', 30, 50);

INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (3, 'PowerPlay Mouse Pad', 10, 30);

INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (4, 'XCD earphones', 10, 60);

INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (5, 'SADES earphones', 12, 30);

INSERT INTO Product (Prod_id, Prod_name, Prod_price, QuantityOnHand)
    VALUES (6, 'PowerPlay Memory Card', 15, 59);

select * from product;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 6 in 0.025 seconds

	PROD_ID	PROD_NAME	PROD_PRICE	QUANTITYONHAND
1	1	PowerPlay Gaming Mouse	20	20
2	2	Logitech Keyboard	30	50
3	3	PowerPlay Mouse Pad	10	30
4	4	XCD earphones	10	60
5	5	SADES earphones	12	30
6	6	PowerPlay Memory Card	15	59

7.2 Insert Data using Procedure

```sql
create or replace PROCEDURE addNewProd(
    v_prod_id IN product.prod_id%TYPE,
    v_prod_name IN product.prod_name%TYPE,
    v_prod_price IN product.prod_price%TYPE,
    v_qty_on_hand IN product.quantityonhand%TYPE
) IS
BEGIN
    INSERT INTO product
        VALUES (v_prod_id, v_prod_name, v_prod_price, v_qty_on_hand);
    dbms_output.put_line('insert successfully');
END addNewProd;

BEGIN
  addNewProd(7, 'beurer thermometer' ,35 ,30 );
END;
```

Script Output ×

Task completed in 0.044 seconds

```
Procedure ADDNEWPROD compiled


PL/SQL procedure successfully completed.
```

7.3 View Product Data to Confirm Procedure Works Well:

	PROD_ID	PROD_NAME	PROD_PRICE	QUANTITYONHAND
1	1	PowerPlay Gaming Mouse	20	20
2	2	Logitech Keyboard	30	50
3	3	PowerPlay Mouse Pad	10	30
4	4	XCD earphones	10	60
5	5	SADES earphones	12	30
6	6	PowerPlay Memory Card	15	59
7	7	beurer thermometer	35	30

8. Procedure to Add a New Sale

- Create a sequence for order_num in Table SalesOrder to generate order_num automatically

```
create sequence order_num_seq
    start with 1
    increment by 1;
```

Script Output

Task completed in 4.349 secor

```
Sequence ORDER_NUM_SEQ created.
```

- Procedure to Add a New Sale

```
-----Procedure to record a new Sale------
create or replace PROCEDURE RecordNewSale
IS
    v_order_num salesorder.order_num%TYPE;
    v_sale_date salesorder.sale_date%TYPE;
    v_sale_total salesorder.sale_total%TYPE;
    v_cust_id salesorder.cust_id%TYPE;
    v_prod_id orderline.prod_id%TYPE;
    v_line_qty orderline.quantity%TYPE;
    v_sale_price orderline.sale_price%TYPE;
    v_line_count number;
    v_start number := 1;
    is_cust_exist number;

    --may use if the customer id does not exist, the user needs to input new customer information--
    v_cust_name customer.cust_name%TYPE;
    v_cust_add customer.cust_add%TYPE;
    v_phone customer.phone%TYPE;
    v_email customer.email%TYPE;
BEGIN
    v_cust_id := &cust_id;

    /*referential integrity checks
    If the customer id does not exist, this program requires the user to creater customer
    information first, then can record a new sale*/
    SELECT count(*) INTO is_cust_exist from customer where cust_id=v_cust_id;

    IF is_cust_exist <> 1 THEN
        v_cust_name := '&customer_name';
        v_cust_add := '&customer_address';
        v_phone := '&customer_phone';
        v_email := '&customer_email';

        INSERT INTO Customer
            VALUES (v_cust_id,v_cust_name,v_cust_add,v_phone,v_email);
        dbms_output.put_line('Customer ID: '||v_cust_id||' Customer Name: '||v_cust_name||' Customer Address: '
        ||v_cust_add||' Customer Phone: '||v_phone||' Customer Email: '||v_email||' insert successfully');
    END IF;

    v_sale_date := sysdate;
    v_sale_total := &total_sale_price;
    v_line_count := &num_of_orderline;
    v_prod_id := &prod_id;
    v_sale_price := &prod_sale_price;
    v_line_qty := &qty_of_this_line;

    INSERT INTO SalesOrder
        VALUES (order_num_seq.nextval,v_sale_date,v_sale_total,v_cust_id);

    --select the order_num just created
    SELECT Max(order_num) INTO v_order_num FROM SalesOrder;

    dbms_output.put_line('SalesOrder Table inserted with Order Number: '||v_order_num||' Sale Date: '||v_sale_date||' Sale Total: '
        ||v_sale_total||' Customer ID: '||v_cust_id||' successfully');

    for iterator in 1 .. v_line_count
    loop
        dbms_output.put_line('There are '||v_line_count||' order lines in this order.');
        dbms_output.put_line('Please insert number '||v_start||' order line information: ');

        v_start := v_start+1;

        INSERT INTO OrderLine
            VALUES (v_order_num,v_prod_id,v_line_qty,v_sale_price,v_line_qty*v_sale_price);
        dbms_output.put_line('OrderLine Table inserted with Order Number: '||v_order_num||' Product ID: '||v_prod_id
            ||' Quantity: '||v_line_qty||' Sale Price: '||v_sale_price||' Line total: '
            ||v_line_qty*v_sale_price||' successfully');

        v_prod_id := &prod_id;
        v_sale_price := &prod_sale_price;
        v_line_qty := &qty_of_this_line;
    end loop;
    commit;
END RecordNewSale;
```

- Procedure compiled

```
Script Output
Task completed in 55.885 seconds
Procedure RECORDNEWSALE compiled
```

- Run the procedure to record a new sale with two orderliness

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Order Number: 21 Sale Date: 24/08/21 Sale Total: 100 Customer ID: 2 insert into SalesOrder Table successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Order Number: 21 Product ID: 2 Quantity: 1 Sale Price: 60 Line total: 60 insert into OrderLine Table successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Order Number: 21 Product ID: 1 Quantity: 1 Sale Price: 40 Line total: 40 insert into OrderLine Table successfully
Process exited.
Disconnecting from the database PDD.
```

- View SalesOrder and OrderLine Table to confirm procedure works well

```
select * from salesorder;
select * from orderline;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.011 seconds

	ORDER_NUM	SALE_DATE	SALE_TOTAL	CUST_ID
1	21	24/08/21	100	2

	ORDER_NUM	PROD_ID	QUANTITY	SALE_PRICE	LINE_TOTAL
1	21	2	1	60	60
2	21	1	1	40	40

9. Trigger to preload data in Despatch Table

- Create a sequence for desp_id in Table Despatch to generate desp_id automatically

```
create sequence desp_id_seq
    start with 1
    increment by 1;
```

Script Output

Task completed in 0.073 seconds

```
Sequence DESP_ID_SEQ created.
```

- Create trigger to preload data as soon as the orderline are saved.

```
create or replace TRIGGER preloadDespatch
AFTER INSERT ON OrderLine
FOR EACH ROW
DECLARE
    v_prod_name product.prod_name%TYPE;
BEGIN

    SELECT prod_name INTO v_prod_name FROM product
        WHERE prod_id = :new.prod_id;

    INSERT INTO despatch (desp_id, order_num, prod_id, prod_name, quantity)
        VALUES (desp_id_seq.nextval, :new.order_num, :new.prod_id, v_prod_name, :new.quantity);

    dbms_output.put_line('Despatch Table preloaded with Order Number: '||:new.order_num||
        ' Product ID: '||:new.prod_id||' Product Name: '||v_prod_name||' Quantity: '
        ||:new.quantity||' successfully');

END preloadDespatch;
```

Script Output | Query Result

Task completed in 0.09 seconds

```
Trigger PRELOADDESPATCH compiled
```

- Run and Test the trigger successfully

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 12 Sale Date: 24/08/21 Sale Total: 12 Customer ID: 1 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 12 Product ID: 1 Product Name: PowerPlay Gaming Mouse Quantity: 1 successfully
OrderLine Table Order Number: 12 Product ID: 1 Quantity: 1 Sale Price: 12 Line total: 12 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 12 Product ID: 2 Product Name: Logitech Keyboard Quantity: 1 successfully
OrderLine Table Order Number: 12 Product ID: 2 Quantity: 1 Sale Price: 21 Line total: 21 successfully
Process exited.
Disconnecting from the database PDD.
```

10. Procedure to Update Despatch Table

- Create procedure for the warehouse staff to update shelf location

```
create or replace PROCEDURE UpdateDespatch(
    v_desp_id despatch.desp_id%TYPE,
    v_prod_id despatch.prod_id%TYPE,
    v_shelfloc despatch.shelfloc%TYPE
) IS
BEGIN
    UPDATE despatch SET shelfloc = v_shelfloc
    WHERE prod_id = v_prod_id
    AND desp_id = v_desp_id;

    dbms_output.put_line('Despatch Table Shelf Location: '||v_shelfloc||' where Product ID: '||
        v_prod_id||' Despatch ID: '||v_desp_id||' updated successfully');

    commit;
END UpdateDespatch;
```

```
Script Output x   Query Result x
Task completed in 0.103 seconds

Procedure UPDATEDESPATCH compiled
```

- Run and Test the procedure successfully

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: U-2 where Product ID: 2 Despatch ID: 6 updated successfully
Process exited.
Disconnecting from the database PDD.
```

11. Test the Procedures

- Delete the random data

```
delete from orderline;
delete from salesorder;
delete from despatch;
```

Script Output | Query Result

Task completed in 0.051 seconds

```
2 rows deleted.

2 rows deleted.

1 row deleted.
```

- Test record new sales procedures & preload despatch trigger with meaningful data

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 1 Sale Date: 24/08/21 Sale Total: 50 Customer ID: 2 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 1 Product ID: 1 Product Name: PowerPlay Gaming Mouse Quantity: 1 successfully
OrderLine Table inserted with Order Number: 1 Product ID: 1 Quantity: 1 Sale Price: 20 Line total: 20 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 1 Product ID: 2 Product Name: Logitech Keyboard Quantity: 1 successfully
OrderLine Table inserted with Order Number: 1 Product ID: 2 Quantity: 1 Sale Price: 30 Line total: 30 successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 2 Sale Date: 24/08/21 Sale Total: 50 Customer ID: 6 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 2 Product ID: 2 Product Name: Logitech Keyboard Quantity: 1 successfully
OrderLine Table inserted with Order Number: 2 Product ID: 2 Quantity: 1 Sale Price: 30 Line total: 30 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 2 Product ID: 3 Product Name: PowerPlay Mouse Pad Quantity: 2 successfully
OrderLine Table inserted with Order Number: 2 Product ID: 3 Quantity: 2 Sale Price: 10 Line total: 20 successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 3 Sale Date: 24/08/21 Sale Total: 30 Customer ID: 3 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 3 Product ID: 3 Product Name: PowerPlay Mouse Pad Quantity: 1 successfully
OrderLine Table inserted with Order Number: 3 Product ID: 3 Quantity: 1 Sale Price: 10 Line total: 10 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 3 Product ID: 4 Product Name: XCD earphones Quantity: 2 successfully
OrderLine Table inserted with Order Number: 3 Product ID: 4 Quantity: 2 Sale Price: 10 Line total: 20 successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 4 Sale Date: 24/08/21 Sale Total: 22 Customer ID: 4 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 4 Product ID: 4 Product Name: XCD earphones Quantity: 1 successfully
OrderLine Table inserted with Order Number: 4 Product ID: 4 Quantity: 1 Sale Price: 10 Line total: 10 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 4 Product ID: 5 Product Name: SADES earphones Quantity: 1 successfully
OrderLine Table inserted with Order Number: 4 Product ID: 5 Quantity: 1 Sale Price: 12 Line total: 12 successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
SalesOrder Table inserted with Order Number: 5 Sale Date: 24/08/21 Sale Total: 39 Customer ID: 1 successfully
There are 2 order lines in this order.
Please insert number 1 order line information:
Despatch Table preloaded with Order Number: 5 Product ID: 5 Product Name: SADES earphones Quantity: 2 successfully
OrderLine Table inserted with Order Number: 5 Product ID: 5 Quantity: 2 Sale Price: 12 Line total: 24 successfully
There are 2 order lines in this order.
Please insert number 2 order line information:
Despatch Table preloaded with Order Number: 5 Product ID: 6 Product Name: PowerPlay Memory Card Quantity: 1 successfully
OrderLine Table inserted with Order Number: 5 Product ID: 6 Quantity: 1 Sale Price: 15 Line total: 15 successfully
Process exited.
Disconnecting from the database PDD.
```

- Test update despatch procedure with meaningful data

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-A where Product ID: 1 Despatch ID: 1 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-B where Product ID: 2 Despatch ID: 2 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-B where Product ID: 2 Despatch ID: 3 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-C where Product ID: 3 Despatch ID: 4 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-C where Product ID: 3 Despatch ID: 5 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-D where Product ID: 4 Despatch ID: 6 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-D where Product ID: 4 Despatch ID: 7 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-E where Product ID: 5 Despatch ID: 8 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-E where Product ID: 5 Despatch ID: 9 updated successfully
Process exited.
Disconnecting from the database PDD.
```

```
Running: IdeConnections%23PDD.jpr - Log
Connecting to the database PDD.
Despatch Table Shelf Location: S-F where Product ID: 6 Despatch ID: 10 updated successfully
Process exited.
Disconnecting from the database PDD.
```

- View SalesOrder Table after populating data with procedure

	ORDER_NUM	SALE_DATE	SALE_TOTAL	CUST_ID
1	3	24/08/21	30	3
2	1	24/08/21	50	2
3	2	24/08/21	50	6
4	4	24/08/21	22	4
5	5	24/08/21	39	1

- View OrderLine Table after populating data with triggers and procedure

	ORDER_NUM	PROD_ID	QUANTITY	SALE_PRICE	LINE_TOTAL
1	2	3	2	10	20
2	1	1	1	20	20
3	1	2	1	30	30
4	3	3	1	10	10
5	2	2	1	30	30
6	3	4	2	10	20
7	4	4	1	10	10
8	4	5	1	12	12
9	5	5	2	12	24
10	5	6	1	15	15

- View Despatch Table after populating data with procedure

	DESP_ID	ORDER_NUM	PROD_ID	PROD_NAME	QUANTITY	SHELFLOC
1	4	2	3	PowerPlay Mouse Pad	2	S-C
2	1	1	1	PowerPlay Gaming Mouse	1	S-A
3	2	1	2	Logitech Keyboard	1	S-B
4	5	3	3	PowerPlay Mouse Pad	1	S-C
5	3	2	2	Logitech Keyboard	1	S-B
6	6	3	4	XCD earphones	2	S-D
7	7	4	4	XCD earphones	1	S-D
8	8	4	5	SADES earphones	1	S-E
9	9	5	5	SADES earphones	2	S-E
10	10	5	6	PowerPlay Memory Card	1	S-F