bjy5305

Physical Database Design

Assignement 1 – Nikkolas Diehl – 16945724

Business Scenario:

“Hotel Stay Longer” is a chain of hotels offering reasonably priced accommodations for customers at their hotels. They have been in business for 5 years, with a branch in each of the 14 cities throughout New Zealand.  
The hotels are all the same design and structure (9 floors, 150 rooms). There are many types of rooms suited for different needs; single, twin, double and suite.  
On average, 1000 reservations are made each day, 60% of which is repeat business.

It is required of the system to migrate to oracle 11g from a proprietary database. This change will occur shortly and be run by the database admin. The following tables will be created for the database:

* Customer
* Reservations
* Room
* Branch
* Invoice

These tables must create following the client’s wishes.

Business Rules:

* The DBA (Database admin) has total control over all tables and table spaces and has un-restricted access.
* The Developer has restricted access to individual tables. He can’t do anything to the invoice table for example because that is automatically generated and is private for the customers. Each table has separate limits for the developer
* Both roles have logins that last the yearly quarter.
* After each yearly quarter the system is cleaned and refreshed

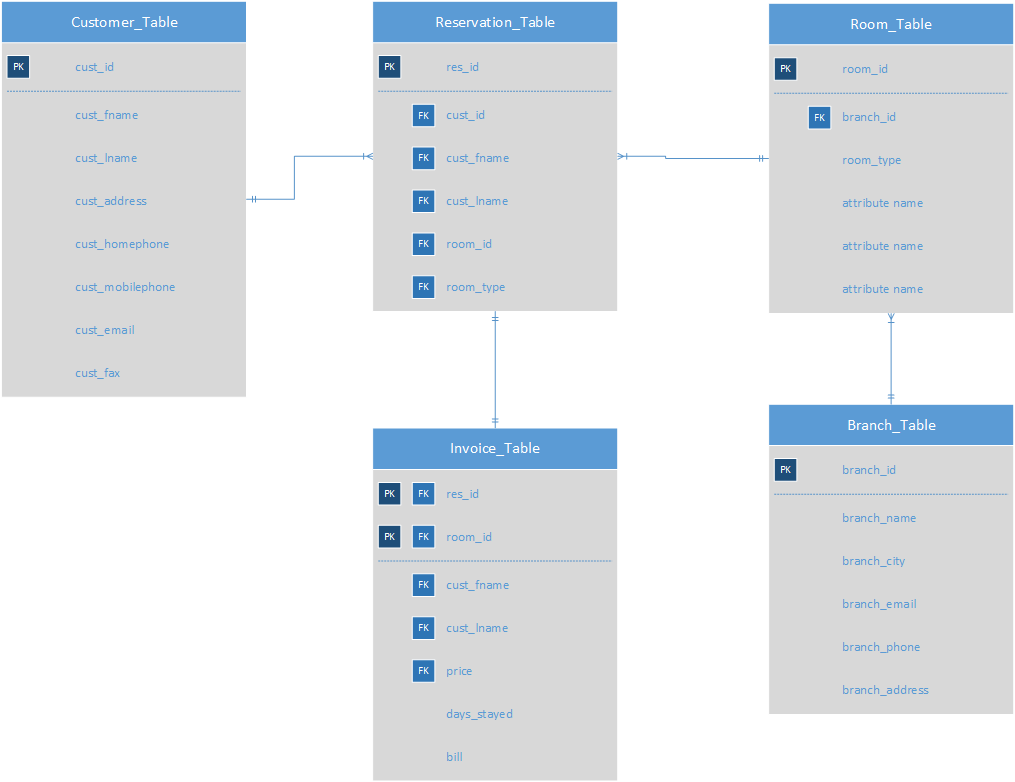
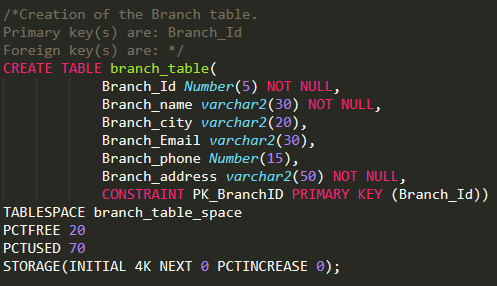
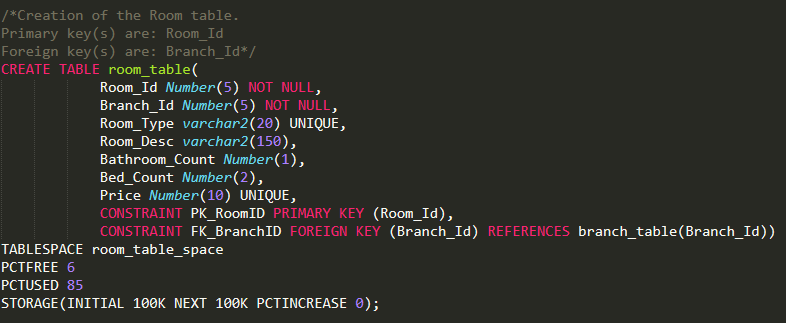
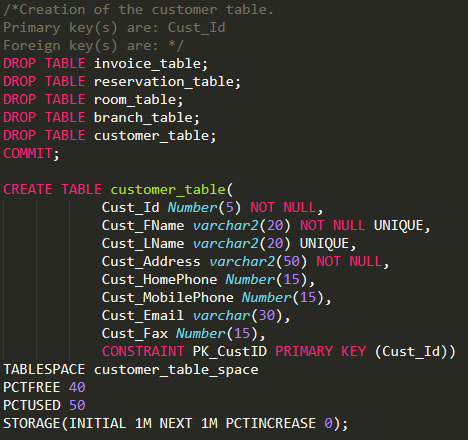
Table Definitions – ER diagram:

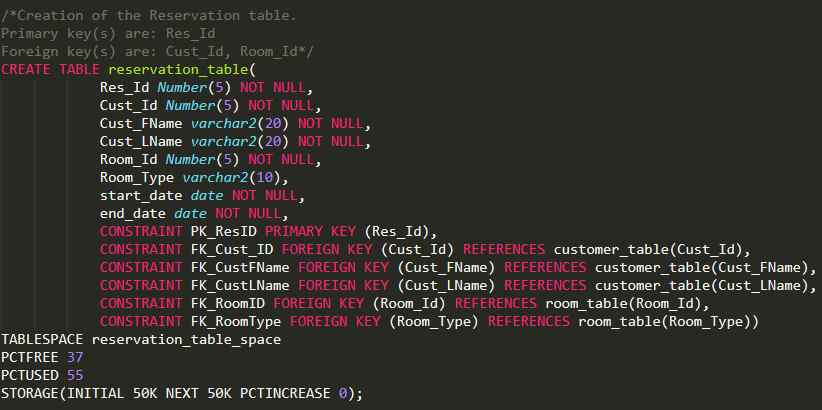
Table Creation Code:

[Link to table creation code](https://pastebin.com/XWM8L656)

Branch Table:  


Room Table:  


Customer Table:  


Reservation Table:  


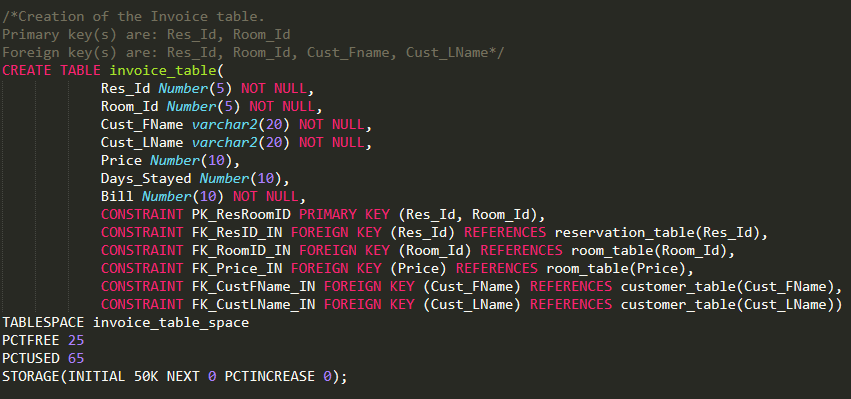
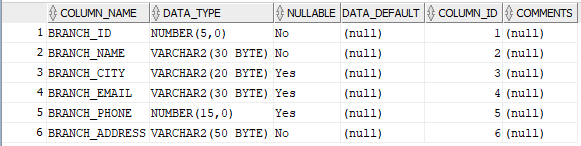
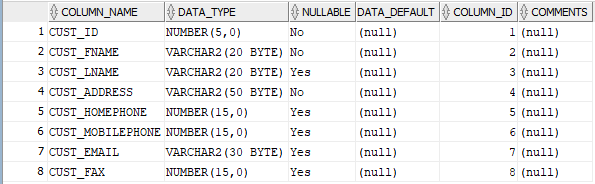
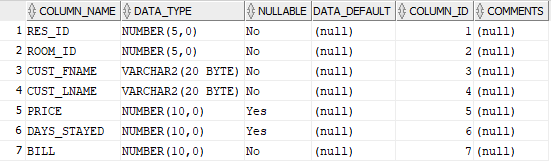
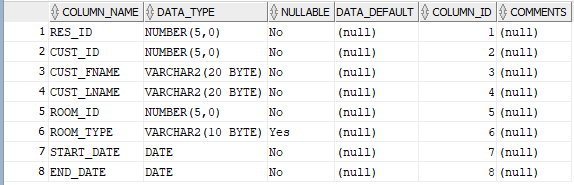
Invoice Table:  


Table Definitions – Table Structure:

Branch Table:  


Customer Table:  


Invoice Table:  


Reservation Table:  


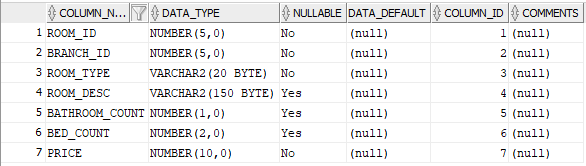
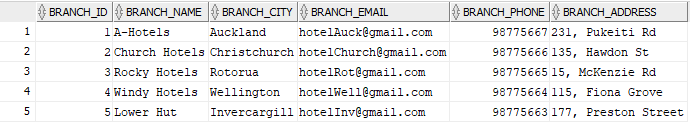
Room Table:  


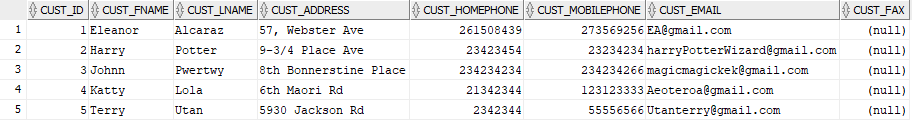
Table Definitions – Data Explanation

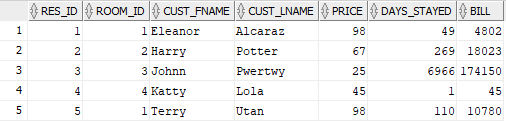
|  |  |  |  |
| --- | --- | --- | --- |
| Row Name: | Data Type: | Constraints: | Reason: |
| Branch Table: |  |  |  |
| Branch ID | Number(5, 0) | Primary Key  Not Null | Branch ID will hold the ID of a given branch, so it is a primary key of any branch plus it cannot be a null value because it is primary.  The data type will give me plenty of possible ID’s to use for all branches. |
| Branch Name | Varchar2(30) | Not Null | Branch Name will hold the name of the given branch. Varchar2(30) data type allows for plenty of possible names. |
| Branch City | Varchar2(20) | -- | Branch City will hold the city location of a branch and can be nullable in case the location cannot be a city. |
| Branch Email | Varchar2(30) | -- | Branch Email will hold the email contact detail of the given branch and can be nullable in case the branch is non-contactable through email. |
| Branch Phone | Number(15, 0) | -- | Branch Phone will hold the phone number of the given branch and is nullable in case the branch is non-contactable through the phone. 30 integers should allow for any length phone number. |
| Branch Address | Varchar2(50) | Not Null | Branch address will hold the address of a given branch and is a varchar2(120) to allow all possible names for an address.  This value is non nullable as a branch must have an address to be contactable in some way. |
| Customer Table: |  |  |  |
| Cust ID | Number(5,0) | Primary key  Not Null | Customer ID will hold the ID of a given customer, so it is a primary key of any customer plus it cannot be a null value because it is a primary.  The data type will give me plenty of possible ID’s to use for all customers. |
| Cust Fname | Varchar2(20) | Not Null | Cust FName will hold the first name of the given customer, so it can’t be null. The data type should be adequate to hold all possible names. |
| Cust Lname | Varchar2(20) | -- | Cust LName will hold the last name of the given customer. This variable is allowed to be null since there exists; many cultures that don’t use a last name. The data type should, again, be adequate to hold all possible last names. |
| Cust Address | Varchar2(50) | Not Null | Cust Address will hold the resident address of the given customer. This variable cannot be null as every customer needs at least the address as a contactable location. |
| Cust Homephone | Number(15, 0) | -- | Cust Homephone will hold the customer home phone number if applicable. It doesn’t have any restrictions because a customer might not have a home phone |
| Cust Mobilephone | Number(15, 0) | -- | Cust Mobilephone will hold the customer mobile phone number if applicable. This can be null |
| Cust Email | Varchar2(30) | -- | Cust Email will hold the customer email if applicable. This is allowed to be null |
| Cust Fax | Number(15, 0) | -- | Cust Fax will hold the customer fax number if applicable. This is allowed to be null as most modern people don’t use fax |
| Invoice Table: |  |  |  |
| Res ID | Number(5, 0) | Composite Primary Key  Foreign Key  Not Null | Res ID contains the reservation ID of a given customer in a given room. This cannot be null and is a composite primary key with the room ID for the invoice table. It is also a foreign key pulling data from the reservation table |
| Room ID | Number(5, 0) | Composite Primary Key  Foreign Key  Not Null | Room ID contains the room ID for a given customer reservation. This cannot be null and is a composite primary key with the res ID for the invoice table. It is also a foreign key pulling data from the room table |
| Cust Fname | Varchar2(20) | Not Null | Cust FName is a foreign key pulling data from the customer table. This cannot be null |
| Cust Lname | Vharchr2(20) | Not Null | Cust LName is a foreign key pulling data from the customer table. This cannot be null |
| Price | Number(10, 0) | Foreign Key | Price holds the price of the room itself and pulls data from the room table |
| Days Stayed | Number(10, 0) | -- | Days stayed is a variable only found in the invoice table and holds the amount of days a customer stays for a reservation. This is used later in a calculation to find the bill for the customer |
| Bill | Number(10s, 0) | Not Null | Bill is a variable only found in the invoice table and is a calculation that multiplies the price of the room per day by the days stayed in the reservation. |
| Reservation Table: |  |  |  |
| Res ID | Number(5, 0) | Primary Key  Not Null | Res ID will hold the reservation ID of a given customer and is use to align a single multi stepped reservation to a single resident. |
| Cust ID | Number(5, )0 | Foreign Key  Not Null | Cust ID will hold the customer ID of a given customer and is pulling data from the customer table |
| Cust FName | Varchar2(20) | Foreign Key  Not Null | Cust FName will hold the first name of a given customer and is pulling data from the customer table |
| Cust LName | Vharchar2(20) | Foreign Key  Not Null | Cust LName will hold the last name of a given customer and is pulling data from the customer table |
| Room ID | Number(5, 0) | Foreign Key  Not Null | Room ID will hold the room ID of a given room that the customer is reserving and is pulling data from the room table |
| Room Type | Varchar2(10) | Foreign Key  Not Null | Room Type will hold the type of the room that the customer is reserving and is pulling data from the room table. |
| Start Date | Date | Not Null | Start date holds the date variable for when the customer starts the reservation |
| End Date | Date | Not Null | End date holds the date variable for when the customer ends the reservation |
| Room Table: |  |  |  |
| Room ID | Number(5, 0) | Primary Key  Not Null | Room ID will hold the ID of a given room and will be used for other tables to reference the room which is why it is a primary key of the room table and Not Null |
| Branch ID | Number(5, 0) | Foreign Key  Not Null | Branch ID will hold the ID of the branch that the given room exists in and will be used as a hierarchical reference to the room. This is why it’s a foreign key and can’t be null. |
| Room Type | Varchar2(10) | Not Null | Room type is a very short description or a single word that describes the room’s type. This cannot be null |
| Room Desc | Varchar2(80) | -- | This is a short to mid length description of the room and can be null if the room type variable gives enough information |
| Price | Number(5, 0) | Not Null | This is the price per day of the room and cannot be null. This is referenced in multiple other tables for calculations |
| Bed Count | Number(1, 0) | -- | Bed Count is a very small number variable that holds the number of beds in the given room. 2 Bytes should be enough to count how many bathrooms there are as it allows up to 99 bathrooms for one room. |
| Bathroom Count | Number(1, 0) | -- | Bathroom Count is also a very small number variable that holds the number of bathrooms in a given room. Again, 2 bytes is enough for this count |

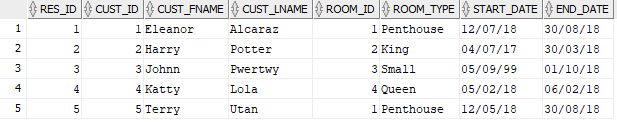
Data Examples:

[Link to data insertion code](https://pastebin.com/AjXYj1Ly)

Branch Table:  


Customer Table:  


Invoice Table:  


Reservation Table:  


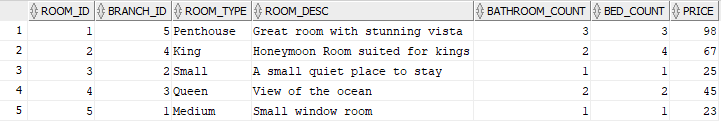
Room Table:   


Table Size Calculations:

[**Link to code to get size values**](https://pastebin.com/G1vAJmw7)

**DB\_BLOCK\_SIZE***:* 8192  
**KCBH**: 20  
**UB4**: 4  
**KTBBH**: 48  
**INITRANS**: 1  
**KTBIT**: 24  
**KDBH**: 14

**Hsize** = DB\_BLOCK\_SIZE - KCBH - UB4 - KTBBH - (INITRANS - 1) \* KTBIT – KDBH  
**Hsize** = 8106

Table Sizes:

* **Branch Table**:
  + 150 Bytes - max per row
  + As stated:
    - **14 branches \* 150 Bytes per branch = 2100 Bytes (~3Kb) table space**
      * Due to the minimum table space being 81 KB, the datafile for a branch specific table space will have to be at minimum, 81 Kb
    - In the branch table, the only thing that might change only a few times every 5 years is the branch email. Branch email is **30 Bytes** **max** needed for updating
    - PCTFree for updating is then **30 Bytes update space/150 Bytes total table = 20% PCTFree**
  + In the branch table, not much of it will ever need to update since most of the data would only be input once when a new branch is created, or deleted when the branch is taken down but the data will mostly remain the same. The only thing that may update is either the email or phone but very, very rarely for the later.
  + **PCTFree = 20%**
  + For storage, but to make sure the PCTUsed is not too high, but enough to store, make PCTUsed: **70%**
* **Customer Table**:
  + 170 Bytes **–** max per row
  + As stated: 12,000 customers per quarter.
    - **12,000 customers \* 170 Bytes per customer = 2,040,000 Bytes (2,040Kb, ~3Mb) table space**
    - In the customer table, there are quite a few things that could update. Customer address, mobile phone and email all have a chance of upating. Customer address is **50 Bytes max**, customer mobile phone is **15 Bytes max**, and customer email is **30 Bytes max**. A total of **95 Bytes max** needed for updating
    - PCTFree for updating is then **95 Bytes update space/170 Bytes total table = 55% PCTFree** however not all of these values might change at the same time, so PCTFree can safely be **40%**
  + In the customer table, there isn’t much that needs to update, but the customer data is holding information on people who are known to change often. The columns that may need to change often or at least once every while are: The customer address, customer mobile phone and customer email.
  + **PCTFree = 40%**
  + For storage, but to make sure the PCTUsed is not too high, but enough to store, make PCTUsed: **50%**
* **Invoice Table**:
  + 80 Bytes – max per row
  + As stated: There are at maximum, 1000 customers on the system every day. Of those, 600 or 60% stay overnight. We can then assume ~400 customers finish their stay each day and thus get an invoice created for them. Invoices are deleted as soon as they are paid, and in most cases, we can assume customers pay immediately when the leave. This means, at maximum with overhead, we need **600 table rows** in the invoice table.
    - **600 invoices in the table \* 80 Bytes per invoice = 48,000 Bytes (~48Kb) table space**
      * Due to the minimum table space being 81 KB, the datafile for an invoice specific table space will have to be at minimum, 81 Kb
    - In the invoice table, as stated by the document, the reservation ID, room ID, customer first name, customer last name and price are all preloaded. That means when a customer ends his/her stay, the invoice is auto updated with the days stayed and bill (which would have originally been set to 0 to avoid the NOT NULL constraint). Days stayed is **10 max** and the bill is **10 Bytes max**. A total of **20 Bytes max** is needed for updating.
    - PCTFree for updating is then **20 Bytes update space/80 Bytes total table = 25% PCTFree**
  + In the invoice table, nothing should change or update. The invoice table is almost entirely pulling data from other tables so when the other tables update or change, this table needs to update. The only things that might need updating in the invoice table are: Days stayed
  + **PCTFree = 25%**
  + For storage, but to make sure the PCTUsed is not too high, but enough to store, make PCTUsed: **6s5%**
* **Reservation Table**:
  + 79 Bytes – max per row
  + As stated: there is about 1000 customers per day so 2000 reservation rows should be enough to cover all reservations made each day. After a reservation is made, started and eventually ended, it is assumed that the reservation row is deleted. That means that after the ~1000 reservations are completed, they are deleted. Since this constantly happens on rotation, with the overhead of 2000 reservation rows, it is easily assumed all ~1000 reservations per day will be cleared before the table can get anywhere near full
    - **2000 max possible reservations at any point in the day \* 79 Bytes max per reservation = 158,000 Bytes (158Kb) table space.**
    - In the reservation table, quite a lot can change. The Room ID can change if the customer changes his/her mind about the room; so can the room type, start date and end date. Room ID is **5 Bytes max**, room type is **10 Bytes max**, start date is **7 Bytes max**, and end date is **7 Bytes max**. A total of **29 Bytes max** will be used for updating.
    - PCTFree for updating is then **29 Bytes update space/79 Bytes total table = 37% PCTFree**
  + In the reservation table, quite a few things will often update as the reservation might be constantly changing. Room ID might update due to the customer changing his mind on his room choice and both the start and end date might be updating quite a lot before the reservation actually begins as the customer is planning.
  + **PCTFree = 37%**
  + For storage, but to make sure the PCTUsed is not too high, but enough to store, make PCTUsed: **55%**
* **Room Table**:
  + 193 Bytes – max per row
  + As stated in the document, there is 150 rooms per branch. With 14 branches, that is 14\*150 = 2100 rooms in total.
    - **2100 max possible rooms \* 193 Bytes per room = 405,300 Bytes (~406Kb) table space**
    - In the room table, almost nothing except price can change unless the entire branch changes and updates it’s rooms which has an almost 0 chance of happening due to the fact it hasn’t happened in the last 5 years of business. Price is **10 Bytes max** so a total of **10 Bytes max** will be used for updating.
    - PCTFree for updating is then **10 Bytes update space/193 Bytes total table = 5.2% PCTFree (6% rounded up)**
  + The room table shouldn’t need much updates since it just holds information on the room itself. The only thing that might change a tiny bit; possibly once a year as a sale, is the price.
  + **PCTFree = 6%**
  + For storage, but to make sure the PCTUsed is not too high, but enough to store, make PCTUsed: **85%**

Table Space Sizes:

**HSIZE**: 8106  
**KDBT**: 4

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table: | PCTFree: | Calculation: | Row Space: | Num of rows in Data block: |
| Branch Table: | 20% | (8106\*(1-20/100))-4 = **6481 Bytes** | 150 Bytes | 6481/150 = 43 rows |
| Customer Table: | 40% | (8106\*(1-40/100))-4 = **4860 Bytes** | 170 Bytes | 4860/170 = 28 rows |
| Invoice Table: | 25% | (8106\*(1-25/100))-4 = **6076 Bytes** | 80 Bytes | 6076/80 = 76 rows |
| Reservation Table: | 37% | (8106\*(1-37/100))-4 = **5103 Bytes** | 79 Bytes | 5103/79 = 64 rows |
| Room Table: | 6% | (8106\*(1-6/100))-4 = **7616 Bytes** | 193 Bytes | 7616/193 = 39 rows |
| Table: | **Rows in Block:** | **Number of Rows Roughly Needed:** | **Number of Blocks Calculation:** | **Number of Blocks:** |
| Branch Table: | 43 | 14 – Branches | CEIL (14/43) | 1 |
| Customer Table: | 28 | 12,000 – Customers | CEIL (12,000/28) | 429 |
| Invoice Table: | 76 | 600 – Invoices | CEIL (600/76) | 8 |
| Reservation Table: | 64 | 2000 – Reservations | CEIL (2000/64) | 32 |
| Room Table: | 39 | 2100 – Rooms | CEIL (2100/39) | 54 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Table: | Total Size: | PCTFree: | PCTUsed: | Initial: | Next: | PCTIncrease: |
| Branch Table: | 2100 Bytes (3Kb) | 20% | 70% | 4K | 0 (only need one block so only one extent needed) | 0 |
| Customer Table: | 2,040,000 Bytes (3Mb) | 40% | 50% | 1M | 1M (3 extents needed) | 0 |
| Invoice Table: | 48,000 Bytes (48Kb) | 25% | 65% | 50K | 0 (all invoices within a day will easily fit within one extent) | 0 |
| Reservation Table: | 158,000 Bytes (158Kb) | 37% | 55% | 50K | 50K (4 extents needed) | 0 |
| Room Table: | 405,300 Bytes (406Kb) | 6% | 85% | 100K | 100K (5 extents needed) | 0 |

Table Space Creation Code:

[Link to tablespace creation code](https://pastebin.com/7TgWRVNa)

User & Roles:

**Policy**:   
The hotel reservation system is designed around the roles implemented for security. Customers for example cannot update, view or delete the data, whilst the admins are able to do all of the above.

**Roles**:

* **DBA**: The DBA manages the entire database and makes sure there is no errors, child dependency problems and so on. He/she gets complete access to the entire database
* **Developer**: The developer is the person that inserts new data when necessary and makes sure everything is up to date

Authorisation Tables:

|  |  |  |
| --- | --- | --- |
| DBA: |  |  |
|  | Branch Table (Yes/No) | Justification: |
| Read | Yes | DBA needs access to everything. |
| Insert | Yes | DBA needs access to everything. |
| Modify | Yes | DBA needs access to everything. |
| Delete | Yes | DBA needs access to everything. |
|  | Customer Table (Yes/No) | Justification: |
| Read | Yes | DBA needs access to everything. |
| Insert | Yes | DBA needs access to everything. |
| Modify | Yes | DBA needs access to everything. |
| Delete | Yes | DBA needs access to everything. |
|  | Invoice Table (Yes/No) | Justification: |
| Read | Yes | DBA needs access to everything. |
| Insert | Yes | DBA needs access to everything. |
| Modify |  |  |  |
| Delete | Yes | DBA needs access to everything. |
|  | Reservation Table (Yes/No) | Justification: |
| Read | Yes | DBA needs access to everything. |
| Insert | Yes | DBA needs access to everything. |
| Modify | Yes | DBA needs access to everything. |
| Delete | Yes | DBA needs access to everything. |
|  | Room Table (Yes/No) | Justification: |
| Read | Yes | DBA needs access to everything. |
| Insert | Yes | DBA needs access to everything. |
| Modify | Yes | DBA needs access to everything. |
| Delete | Yes | DBA needs access to everything. |
| Developer: |  |  |
|  | Branch Table (Yes/No): | Justification: |
| Read | Yes | As a developer, he/she needs to be able to read all tables |
| Insert | Yes | If a contractor asks the developer to insert a new branch due to it being constructed, he/she needs to be able to do so |
| Modify | Yes | A developer might need to change the contact details of a branch |
| Delete | No | The developer should not be allowed to delete a branch due to many table constraints that might cause massive havoc. |
|  | Customer Table (Yes/No): | Justification: |
| Read | Yes | As a developer, he/she needs to be able to read all tables |
| Insert | No | Only customers or desk clerks and the like can insert a new account |
| Modify | No | Only customers or desk clerks and the like can modify their own account |
| Delete | No | Only customers or desk clerks and the like can modify their own account |
|  | Invoice Table (Yes/No) | Justification: |
| Read | No | This would be a breach of privacy between the customer and the hotel |
| Insert | No | The developer cannot read the table, so they can’t insert either |
| Modify | No | The developer cannot read the table, so they can’t modify either |
| Delete | No | The developer cannot read the table, so they can’t delete anything from it as well. |
|  | Reservation Table (Yes/No) | Justification: |
| Read | Yes | As a developer, he/she needs to be able to view the reservation table and get the rooms ready |
| Insert | No | Only customers can insert reservations |
| Modify | No | Only customers can modify reservations |
| Delete | No | Only customers can cancel a reservation |
|  | Room Table (Yes/No) | Justification: |
| Read | Yes | As a developer, he/she needs to be able to read the room table for editing |
| Insert | Yes | If a developer gets contracted by someone higher up to create a new room after it has been constructed, the developer needs to do so. |
| Modify | Yes | If he is contracted because some room information has been changed, the developer needs to be able to |
| Delete | No | Once a room, is created, it has many child dependencies that can’t be broken |

Security Policy Matrix:

|  |  |  |
| --- | --- | --- |
| User Type: | System Privileges: | Justification |
| DBA | ALL privileges | To be able to manage and maintain the database system as well as setup and control it, the DBA needs all privileges |
| Developer | On branch table:   * SELECT * INSERT * UPDATE   On customer table:   * SELECT   On reservation table:   * SELECT   On room table:   * SELECT * INSERT * UPDATE   System Privileges:   * create procedure, * create any procedure, * alter any procedure, * drop any procedure, * execute any procedure, * create any index, * alter any index, * drop any index, * create session, * create sequence, * create any sequence, * alter any sequence, * drop any sequence, * select any sequence, * create table, * comment any table, * create trigger, * create any trigger, * alter any trigger, * drop any trigger, * create view, * create any view, * drop any view | To be able to view, edit some things and maintain the system. If the developer is contracted to do so, he can insert new branches or rooms, modify them and so on.  He cannot edit the reservation table or customer table, and cannot even see the invoice table for privacy reasons  Procedure control = A developer needs to be able to create a procedure and control it.  Index Control = A developer needs to be able to create easy to use indexing for the tables.  Session creation = A developer needs to be able to connect to DB  Sequence control = same reason as procedure control  Table creation/commenting = the developer needs to be able to comment on the tables for updates and for development.  Trigger control = same reason as procedure control  View control = for development and work |

Profiles:

Profiles are a named set of resource limits placed upon the user whenever the user accesses the database. I have created two profiles to control the DBA and Developer as they work.

* **AProfile (Admin Profile)**: This profile is assigned to the DBA and is an unlimited session profile
* **DProfile (Developer Profile)**: This profile is assigned to the Developer and has multiple restrictions

|  |  |  |
| --- | --- | --- |
| Controls: | AProfile: | DProfile: |
| Sessions\_per\_user | Unlimited | 2 sessions allowed per user |
| Cpu\_per\_session | Unlimited | 180,000 100th’s of a second. 30 minutes of cpu\_per\_session |
| Cpu\_per\_call | Unlimited | 2000 (20 seconds per call) |
| Idle\_time | 20 minutes of idle before being logged out | 5 minutes of idle before being logged out |
| Connect\_time | Unlimited | 30 minutes before being disconnected |
| Failed\_login\_attempts | 3 times before locking for admin security | 5 times before locking for developer security |
| Password\_lock\_time | 0.04 days – 1-hour lock time | 0.04 – 1-hour lock time |
| Password\_life\_time | 91 days – 1 yearly quarter before system clean | 91 days – 1 yearly quarter before system clean |
| Password\_gace\_time | 5 days to change your password before being reset | 5 days to change your password before being reset |

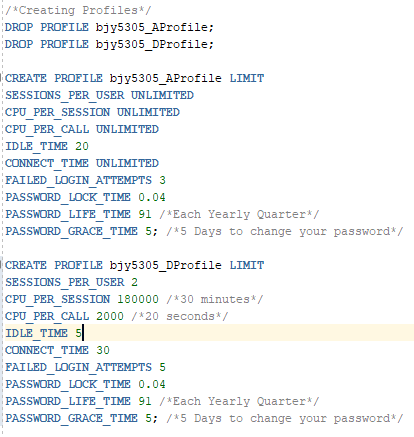
Users:

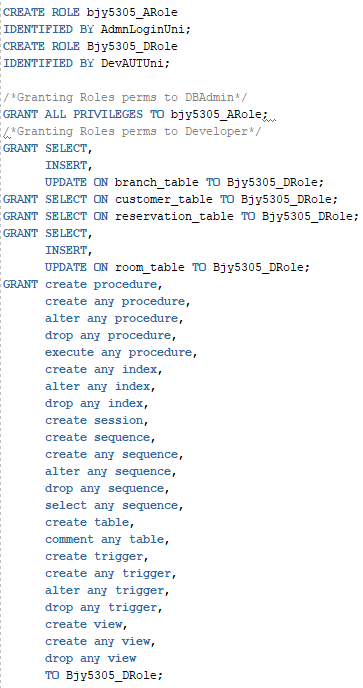
There are currently two different users, although more users being added will need to be granted roles and will need to login using one of the two profiles:

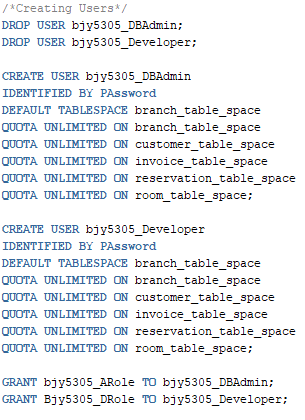
* **DBAdmin**: As an admin, he/she needs unlimited quota on all tablespaces
* **Developer**: As a developer, he/she needs unlimited quota on all tablespaces. Although with the roles granted to him/her, he/she will be restricted on what he can do in each table space

Profile, role and user creation code:

[Link to profile, role and user creation code](https://pastebin.com/LcvUBv5a)

Profiles:  


Roles:  


Users:  
****

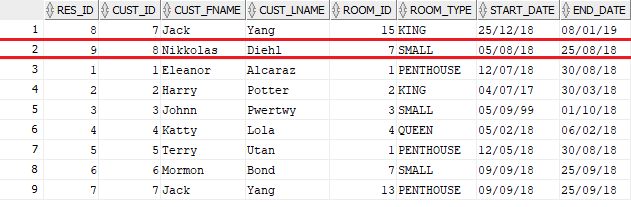
New reservation and invoice script:

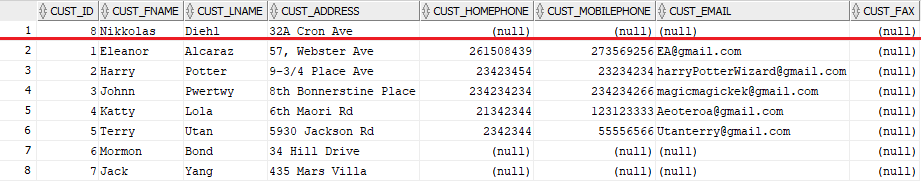
[**Link to new reservation script code**](https://pastebin.com/sqr47d1D)

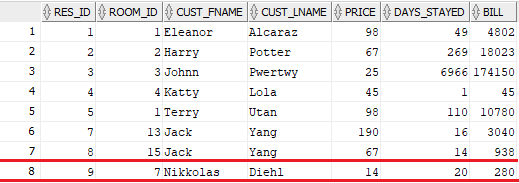
[**Link to new invoice trigger script code**](https://pastebin.com/SDiujMFX)

Example of new reservation script and new invoice trigger script with a new reservation

****

Example of the new reservation in the table:  


Example of the new person, because he didn’t exist before he made a reservation:  


Example of the new invoice created for this person:  


Code:

