Data Integrity

Agenda

- What is Data Integrity
  - Classification
- ACID Properties
- Normalization
- Constraints on Relation
- Referential Integrity

What is Data Integrity?

According to data management community and data Governance,

Integrity of data ensures an appropriate lineage of business entities in the data flow process.

Data Integrity

- Ensures smooth business flow of business.

- It is the key aspect of design of data flow process.

- It ensures the consistency of the data through entire project life cycle. It means the data should be relevant across the business.

What is Data Integrity?

In order to establish the data lineage, organization should

- Receive ***accurate*** data as if the records are not ignored without any flaws. Hence the loss of data costs to organization.

- Consistency of data flow across business processes.

  So, None of the entities(tables) are not struck in the middle of the business process due invalid or irrelevant data.

- Data generated should be reliable within business units. So that the data is consistent when one business unit shares the data with other business unit.

Loss of Data integrity results in

- Audit penalty by data governance.
- Costs associated to unused data maintenance in billions due to additional storage.
- Consuming of additional RAM memory by SQL programs running on the data.
- Purchase of additional disk space either in standalone or cloud environments.
- Analytics may show inappropriate results.

Loss of Data integrity results in

- Quality control is an overhead
- Inappropriate decision making when multiple strategic reports are generated without any quality of data

Data Integrity is classified as:

1. Row level integrity
2. Column level integrity
3. Referential integrity

Row level integrity

- Each row in the table is referenced by unique values, and avoids conflicts with other rows.
- This unique identifier is normally known as primary key of the table

  Example : CUSTOMER table uses single column ***Cust_Id*** as the unique identifier

```
Select * from CUSTOMER where Cust_Id=123001
```

customer_id	customer_name	Address	state_code	Telephone
123001	Oliver	225-5, Emeryville	CA	1897614500

Column level integrity

- Every attribute or Column of a table held a business meaning.

  It consists of metadata means an information of a column. The information of the column explains the meaning and purpose of the column defined in the table and how it is later referenced across the organization.
- In order to achieve this, the data that is stored in a column is adhere to the same format and definition.
- This includes ***data type, size, range*** **of possible values**

  E.g: Cust_Id data-type is integer which accepts only integer values.

Column level integrity

SELECT * FROM EQUIPMENT.

MACHINE_ID	MACHINE_NAME	MODEL
MAC01	SPINNING	2018
MAC02	DYEING	2019

DESC EQUIPMENT

Field	Type	Null	Key
MACHINE_ID	varchar(20)	NO	PRI
MACHINE_NAME	varchar(30)	YES	
MODEL	int(11)	YES	

In this table,
MODEL column will not accept Alphabets because it is defined as an Integer.

INSERT into EQUIPMENT values

('MAC03' , 'SPINNING' , ‘Nineteen’)

Column level integrity - ensures to insert correct values according to its datatype.
So the below statement will successfully insert the record.

INSERT into EQUIPMENT Values ('MAC03' , 'SPINNING' , 2019)

Referential Integrity

Referential Integrity -

Ensures consistency of records between two related tables.

It establishes relationships between two different tables using referencing columns.

It sets up a parent child relationship between two tables.
So the child table always references the parent table.
At the same time, the parent table prevents entering a row in child table.

Establishing a Parent - child relationship between tables is achieved by defining the tables using - FOREIGN KEY and PRIMARY KEY.

Defining Foreign Key to show referential Integrity

doit YOURSELF

```
#Set up a Foreign key
ALTER TABLE LEASE
ADD   FOREIGN KEY (MACHINE_ID )
REFERENCES EQUIPMENT (MACHINE_ID)
```

Field	Type	Null	Key
MACHINE_ID	varchar(20)	YES	MUL
INVOICE	varchar(10)	NO	PRI
QUANTITY	int(11)	YES	
UNIT_PRICE	decimal(12,2)	YES	
LEASE_DATE	date	YES	
LEASE_EXPIRY	date	YES	

MACHINE_ID defined as MUL, means the field can accept duplicate values, but the values should already present in its parent Column in Equipment table

do it YOURSELF

Here Foreign Key **will *not* allow** INSERT or UPDATE a record in the child table for which there is no existing record in parent table.

E.g:

```
INSERT INTO LEASE VALUES
    ( 'MAC04' , 'INV004' ,7 , 20100, '2019-04-01' , '2022-04-01'  ) ,
```

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

Records

Equipment

MACHINE_ID	MACHINE_NAME	MODEL
MAC01	SPINNING	2018
MAC02	DYEING	2019
MAC03	SPINNING	2019
MAC04	SPINNING	2019

LEASE

MACHINE_ID	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC01	INV001	7	20100.00	2019-04-01	2022-04-01
MAC01	INV002	3	35000.00	2020-02-01	2021-02-01
MAC02	INV003	4	15000.00	2019-06-01	2021-06-01
MAC04	INV004	7	20100.00	2019-04-01	2022-04-01

Now let's delete an existing record from EQUIPMENT table and see if its corresponding matching records on child table is automatically deleted or not

```
DELETE from EQUIPMENT
where MACHINE_ID = 'MAC01';
```

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails

Similarly , You can delete a row from Parent table because child row is existing in a table.

ACID Properties

ACID Properties

- A transaction is a single logical unit of work which acts on the database contents
- The database state is in tandem with every transaction to ensure the data integrity
- Hence , in a transaction processing, database application follows the standard four properties to ensure a smooth flow of the business

ACID Properties

- Atomicity
- Consistency
- Isolation

Atomicity

Atomicity:
Complete the data transaction by 100% otherwise retain the original status.

Ex: An account holder - “A” intends to **debit** $1000 from his account and **credit** account holder - “B” for sum of $1000.

An atomicity here to ensures individual accounts are debited and credited successfully.

In case of any transaction failure, retain the original account balances.

Consistency

Consistency:

Consistency of data ensures the data **is not leaked** in any case of success or failure of transactions.

Ex: “A” *debits* **2000***$ to “B”*, but “B” is *credited with* **1000**$ only. Hence, B lost $1000.

Here the transaction is success but not consistent.

Database Application should always ensure the transformation rules are consistent without loss to actual data.

Isolation

Isolation:

Database application ensures every transaction is individual, secured and is hidden from other transactions.

Ex: Transaction of “A” *debits* **2000***$ to “B” is invisible to any others.*

Similarly, transaction of “B” *crediting* **2000**$ from “A” is invisible to any others.

Here the transactions that are debit of “A” and credit of B is invisible and will not let others to interfere and leak the data during transaction and do not provide room for un-ethical hacking.

Isolation

Isolation:

Database application ensures every transaction is individual, secured and is hidden from other transactions.

The transactions are serialized to avoid below conflicts when occurring at same time.

Ex: *Edward* has balance of **$ 5000** but he has given two cheques to *Richard* and *Sharma* for a sum of **$3000 and $2700.**

In reality, Edward’s deficit balance is $700.

Isolation of transactions helps to prevent such malfunctions in real time bank transactions.

Durability

Durability:

Data should be persistent after before and after a transaction.

Ex: Edward has account balance of $4500 and swipes $2000 for purchasing product in an Amazon. All of sudden network goes down and the order is not fulfilled but money is debited.

To ensure durability of data, the database maintains the state of data before and after transaction.
Hence, Edward’s transaction can be successful or it can be reversed.

Normalization

Functional Dependency

All the non-key Columns of a table are said to be dependent on Primary key, which uniquely identifies the rows in a table.

For example:
Consider a CUSTOMER table with attributes:

Cust_Id, Name, Address, State, Telephone.

Field	Key
Cust_id	PRI
Name	
Address	
State_code	
Telephone	

Here **Cust_Id** is a primary Key, which uniquely identifies the remaining columns.

Hence, the non-key columns are said to be functionally dependent on Primary Key.

Normalization

- Normalization is the process of organizing the data attributes with their relationships
- Normalization minimizes the redundancy of data rows
- Normalization minimizes the dependency of columns.
- It eliminates the undesirable characteristics like Insertion, Update and delete of anomalies(flaws).
- Normalization divides the larger table into the smaller tables and establishes entity-relationship among those tables.

Normalization

Normalization is evolved into several stages over a period of time.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully ***functional*** dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no ***transition*** dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no *multivalued* dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

A table is said to be in *First Normal Form*, when it follows the following 4 rules:

1. A column/attribute in a table should consists of a scalar atomic value
2. All the values stored in a column should have same business term
3. All the columns in a table should be represented with unique names
4. The order of rows and columns doesn’t matter

First Normal Form (1NF) - Example

- One of the branches of a bank is piling up with entries by adding multiple branches with different IFSC codes.

  E.g: One city keep opening multiple branches in different areas, and trying finds difficulty in building relationship between columns and rows, and failed to store record of each branch details separately.

- Hence ***1-NF*** **resolves such problems by dec**omposing the multi-valued attribute phone_number in bank_branches table

Normalization before and after 1 - NF

SELECT * from ALL_BANKS /* before 1 - NF , it is multi- valued in one column */

BANK	CITY	BRANCH	IFSC_CODE
HDFC	HYDERABAD	ABIDS, MALAKPET	HDFC0000145, HDFC0000849
SBI	BENGALURU	BTM, Whitefield	SBI00006756, SBI00002311
ICICI	MUMBAI	MALAD, DADAR	ICIC0007645, ICIC0003349

Select *from BANKS

BANK	CITY	BRANCH	IFSC_CODE
HDFC	HYDERABAD	ABIDS	HDFC0000145
HDFC	HYDERABAD	MALAKPET	HDFC0000849
SBI	BENGALURU	BTM	SBI00006756
SBI	BENGALURU	Whitefield	SBI00002311
ICICI	MUMBAI	MALAD	ICIC0007645
ICICI	MUMBAI	DADAR	ICIC0003349

In 1- NF, Multi-values are placed in different rows

Second Normal Form (2NF)

- All non-key & independent attributes are ***fully functionally dependent*** on the primary key.
- 2NF is evolutioned after 1NF , and handled the redundancy of data effectively
- Below rules are followed to achieve 2NF
- In 2NF, the data must be in 1NF

A table before 2-NF

Select * from Equipment_Lease

MACHINE_ID	MACHINE_NAME	MODEL	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC01	SPINNING	2018	INV001	7	20100.00	2019-04-01	2022-04-01
MAC01	SPINNING	2018	INV002	3	35000.00	2020-02-01	2021-02-01
MAC02	DYEING	2019	INV003	4	15000.00	2019-06-01	2021-06-01

In this example, two problems arises while generating Invoice.
Redundancy:
Whenever a new Invoice is generated for a Machine,
a new record is inserted into table by **repeating** machine Name, Machine Id, model Numbers .

Dependency:
Here the other columns related to Invoice are made dependent on machines. Hence there is too much dependency.

Let us see how 2-NF solves the problem

A table before 2-NF

Defining CUSTOMER_details table:

- Candidate/composite key : [MACHINE_ID, INVOICE]
- Lease_date, lease_expiry, Quantity, price columns are *partially* dependent on Candidate key.
- To achieve 2-NF,

  - Split the table into two, split the candidate key into two primary keys.
  - Make the non-key columns *fully* dependent on respective primary keys.

Tables after 2-NF

Desc Equipment

MACHINE_ID	MACHINE_NAME	MODEL
MAC01	SPINNING	2018
MAC02	DYEING	2019

Primary Key:
Machine_ID

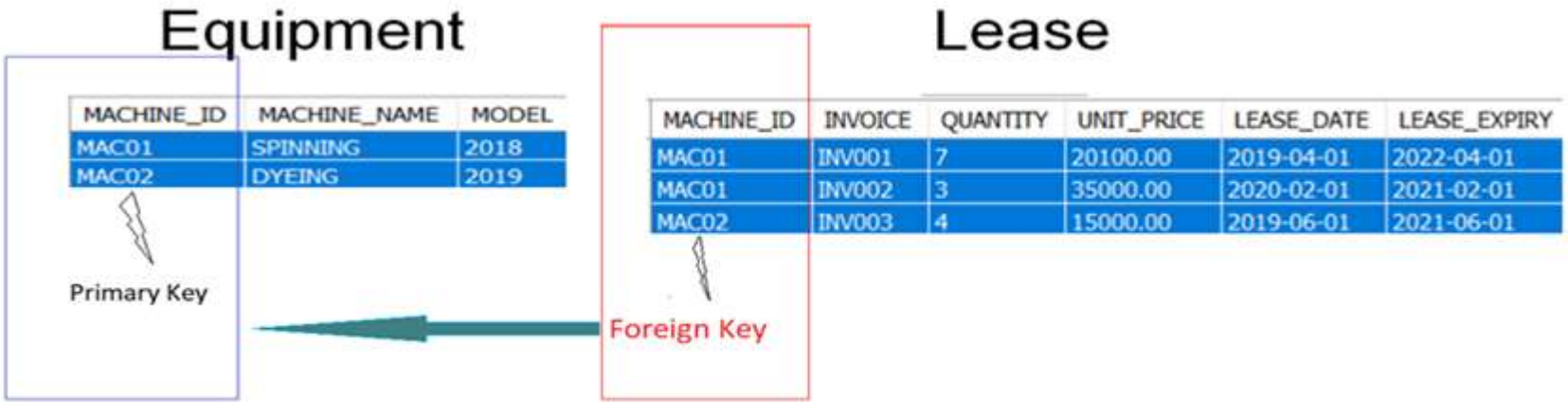
Desc LEASE

INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
INV001	7	20100.00	2019-04-01	2022-04-01
INV002	3	35000.00	2020-02-01	2021-02-01
INV003	4	15000.00	2019-06-01	2021-06-01

Primary Key: Invoice

2-NF with referential integrity

Later after 2 - NF, add referential integrity using Foreign Key and primary key relationship.



Achieved 2- NF normalization.

Third Normal Form (3-NF):

To achieve 3-NF

- The attributes of a table should already be in 2-NF, so full functional dependency was achieved already in 2- NF.
- If any non-prime have transitive dependency, this is resolved with 3 - NF.
- Transitive dependency is like attribute - A depends on attribute - B, then attribute - B depends on attribute - C and so on

ACCOUNT_TRANSACTION : (Before 3- NF)

SELECT * from *ACCOUNT_TRANSACTION*

Cust_Id	Account_Number	Account_type	Balance	Tran_ID	Cheque_No	Pay_purpose	Tran_Amount
123002	4000-1956-2001	SAVINGS	900000	T99901	CHQ0001	Education Fee	500000
123002	4000-1956-2001	SAVINGS	950000	T99907	CHQ0023	Deposit	50000
123002	5000-1700-5001	FIXED DEPOSITS	7500000	T99904	CHQ0002	Overdraft	500000
123002	5000-1700-5001	FIXED DEPOSITS	7950000	T99904	CHQ0006	Annual deposit	450000

In this table,

- Tran_amount , cheque_No , Pay_purpose **dependent on** Tran_id
- Tran_id is **dependent on** Account_Number
- Account_Number is **dependent** on Cust_id
- All non-prime attributes (Tran_amount , cheque_No , Pay_purpose) are **transitively dependent on** super key(**Cust_ID**).

Hence violating the rule of third normal form.

3- NF

In order to achieve 3 - NF,
Split the *ACCOUNT_TRANSACTION* table into two

1. ACCOUNT
2. TRANSACTION

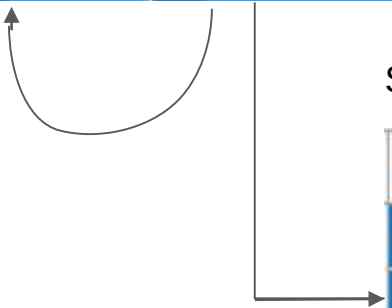
- Eliminate *transitive* dependency (Tran_amount , cheque_No , Pay_purpose) on Cust_Id
- Resolve the *redundancy* of Data by using **3- NF**

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

After 3- NF

Select * from ACCOUNT

Cust_id	Account_Number	Account_type	Balance
123002	4000-1956-2001	SAVINGS	950000
123002	5000-1700-5001	FIXED DEPOSITS	7950000



Select * from Transaction

Account_Number	Tran_ID	Cheque_No	Pay_purpose	Tran_Amount
4000-1956-2001	T99901	CHQ0001	Education Fee	500000
4000-1956-2001	T99907	CHQ0023	Deposit	50000
5000-1700-5001	T99904	CHQ0002	Overdraft	500000
5000-1700-5001	T99904	CHQ0006	Annual deposit	450000

Boyce-Codd Normal form (BCNF)

- BCNF resolves problems related to redundancy of data appeared in 3-NF
- A 3 - NF table is split into DIMENSION and FACT tables and solves the problem of 3 - NF
- 3- NF issue arises when multiple sub-transactions appeared for single transaction.

After BCNF

- **To normalize using BCNF, we will create dimension and facts tables.**
- **The Fact table data consists of unique records. E.g : Account and its corresponding Cheque issued details.**
- **Dimension table consists of break up details of cheque. E.g:**

Dimension:

A cheque is issued for payment of an Education fee. But the cheque is further divided the sum of money into fees, tax , cess amount . All of these are dimensions of single transaction.

Fact :

The high level records consists of cheque Number and its corresponding account.

After BCNF

Select * from TRAN_DETAILS

Account_Number	Tran_ID	Cheque
4000-1956-2001	T99901	CHQ0001
4000-1956-2001	T99907	CHQ0023

Select * from CHEQUE_DETAILS

Cheque	BILL_NO	Pay_purpose	Tran_Amount
CHQ0001	BIL001	Education Fee	400000
CHQ0001	BIL002	Income Tax	100000
CHQ0023	BIL003	Deposit	46000
CHQ0023	BIL004	TDS	4000

Constraints on Single Relation

Constraints on Single Relation

- SQL Constraints are the rules used to limit the type of data that can go into a column/table

- Constraints ensure the accuracy and integrity of the data inside the column/ table

Constraints can be divided into the following two types,

1. Column level constraints: Limits only column level data values
2. Table level constraints: Limits complete table dataset

Constraints on Single Relation

- Following are the most used constraints on Single relationship that are applied to Columns in a single table

  NOT NULL, DEFAULT

Other constraints like below will be discussed later.

- UNIQUE
- PRIMARY KEY
- CHECK

NOT NULL:

- NOT NULL constraint restricts a column from having a NULL value
- After applying NOT NULL constraint to column, you cannot pass a null value into that column

```
EX:  ALTER TABLE BANKS Modify IFSC_CODE varchar(35) NOT NULL;
```

Field	Type	Null
BANK	varchar(20)	YES
CITY	varchar(20)	YES
BRANCH	varchar(20)	YES
IFSC_CODE	varchar(35)	NO

IFSC_CODE is changed its NULL property from “YES” to “NO” .
It means IFSC_CODE value must not be blank.

NOT NULL:

```
insert into BANKS  values
('HDFC', 'HYDERABAD',  'KOTHAPET',  NULL)
```

Column Check constraints

CHECK Constraint

- CHECK constraint conditionally validates the column values before insertion of the record, and prevents from any unwanted entry of record.

```
ALTER table TRANSACTION ADD CHECK (
    abs(Tran_Amount) < 2000000 )   ;
```

- The table ensures that transaction amount is not exceeding more than the 2000000/- for any new entry of transaction.

Try:

```
    Insert into  TRANSACTION  values
    ( '5000-1700-5001', 'T99305' , 'CHQ0022' , 'Cash Withdraw'       , 2000009
) ;
```

Error Code: 3819. Check constraint 'TRANSACTION_CHK_1' is violated.

This file is meant for personal use by vijaykumarsingh93854@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.

Entity Integrity

Entity Integrity

- Entity Integrity ensures the rows are consistently receives the values without nulls and duplicates
- To ensure the entity integrity, a primary key / UNIQUE without NULLS are defined on table columns
- These constraints serves as a unique and non-null identifier for rows in the table

Uniqueness Constraints

Uniqueness Constraints

- UNIQUE constraint ensures a column will only have unique values

  Eg:

```
ALTER TABLE BANKS ADD UNIQUE ( IFSC_CODE );
```

Field	Type	Null	Key
BANK	varchar(20)	YES	
CITY	varchar(20)	YES	
BRANCH	varchar(20)	YES	
IFSC_CODE	varchar(35)	NO	PRI

Along with Not NULL, UNIQUE constraint added to IFSC_CODE will prevent adding duplicate values .

Uniqueness Constraints

Select * from BANKS

BANK	CITY	BRANCH	IFSC_CODE
HDFC	HYDERABAD	ABIDS	HDFC0000145

Try :

```
Insert into BANKS  values
('HDFC', 'HYDERABAD',  'ABIDS'  ,  'HDFC0000145')
```

Error Code: 1062. Duplicate entry 'HDFC0000145' for key 'IFSC_CODE'

PRIMARY KEY:

- Primary key constraint is defined on a column and it uniquely identifies each record in a table

- A Primary Key by default creates an Index on a column. An index key is used for search records based on conditions

```
ALTER TABLE ACCOUNT ADD PRIMARY KEY (Account_Number) ;
```

PRIMARY KEY:

Select * from ACCOUNT

Cust_id	Account_Number	Account_type	Balance
123002	4000-1956-2001	SAVINGS	950000
123002	5000-1700-5001	FIXED DEPOSITS	7950000

Desc ACCOUNT

Field	Type	Null	Key
Cust_id	int(11)	YES	
Account_Number	varchar(50)	NO	PRI
Account_type	varchar(20)	YES	
Balance	int(11)	YES	

Try :

INSERT INTO ACCOUNT VALUES

(123002, '4000-1956-2001', 'SAVINGS', 950000);

Error Code: 1062. Duplicate entry '4000-1956-2001' for key 'PRIMARY'

Referential Integrity Problems

Referential Integrity Problems

- Enforcing referential integrity will cause limitations to its referenced tables on DML operations such as Insert, Delete and Update

- Referential integrity constraints decreases the performance of DML operations as the SQL engine should refers to both the table which is being updated and as well as referenced tables

Referential Integrity problems

- Referential integrities are overhead when exporting the tables from one database to other database
- Problems like foreign key should be disabled and enabled after exporting
- re-enabling foreign keys lost data integrity as it will not scan appropriately

Delete and Update Rules

Delete and Update Rules

- Recapping :-
- Foreign key constraints always references a primary or unique key
- A foreign key value when assigned with NULL doesn’t references any primary key value in another table
- But if foreign key has any valid value, then it must have an associated value in primary key
- During any DML operation on Parent tables , there are different rules that effect on associated values in child tables

Rules on Updating Foreign Key

- **When updating foreign key column with a value that is not present in primary key column, it violates the referential integrity**

Desc EQUIPMENT

Field	Type	Null	Key
MACHINE_ID	varchar(20)	NO	PRI
MACHINE_NAME	varchar(30)	YES	
MODEL	int(11)	YES	

Desc LEASE

Field	Type	Null	Key
MACHINE_ID	varchar(20)	YES	MUL
INVOICE	varchar(10)	NO	PRI
QUANTITY	int(11)	YES	
UNIT_PRICE	decimal(12,2)	YES	
LEASE_DATE	date	YES	
LEASE_EXPIRY	date	YES	

Foreign Key (MUL) in LEASE table is referring Primary Key in EQUIPMENT .

Rules on Updating Foreign Key

- ***Primary Key table:***
  SELECT * FROM EQUIPMENT WHERE MACHINE_ID = 'MAC01'

MACHINE_ID	MACHINE_NAME	MODEL
MAC01	SPINNING	2018

- *Foreign key table*

  `SELECT * FROM LEASE WHERE` **MACHINE_ID = 'MAC01'**

MACHINE_ID	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC01	INV001	7	20100.00	2019-04-01	2022-04-01
MAC01	INV002	3	35000.00	2020-02-01	2021-02-01

Now, try updating the Foreign Key column value

Proprietary content. © Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

Trying to update Foreign key

Try : ***update existing*** **MACHINE_ID = 'MAC01' in LEASE table.**

Update LEASE set MACHINE_ID = **'MAC09'** where MACHINE_ID = 'MAC01'

Rules on Updating Primary Key

- When updating Primary key column with a value that is not present in foreign key column, and violates the referential integrity
- Trying to update Primary Key

  Update EQUIPMENT
  set MACHINE_ID = **'MAC09'** where MACHINE_ID = 'MAC01'

Rules on deleting Primary Key records

- Delete of a record in a parent table that is referenced by child's foreign key columns
- However, delete of a record in child table doesn't effect in parent table
- Trying to delete parent table record that is being referenced by foreign key

Try:

```
Delete from EQUIPMENT where MACHINE_ID = 'MAC01'
```

This file is meant for personal use by vijaykumarsingh93854@gmail.com only.
Sharing or publishing the contents in part or full is liable for legal action.
Proprietary content. ©Great Learning. All rights reserved. Unauthorized use or distribution prohibited.

Cascaded, Deletes and Updates

Cascading:

- To overwrite the DML rules on referential integrity fields, CASCADE is issued on the table having foreign key column
- So that changes to parent table records are automatically reflected in child table

Cascading can be done for two types of DML statements

- UPDATE CASCADE
- DELETE CASCADE

ON UPDATE CASCADE

Before CASCADE

Update EQUIPMENT
set MACHINE_ID = **'MAC09'** where MACHINE_ID = 'MAC01'


```
/* Drop the identified Lease foreign key in the above Error */
 ALTER TABLE LEASE DROP       FOREIGN KEY  lease_ibfk_1 ;
/* Add foreign key again */
 ALTER TABLE LEASE ADD
                        FOREIGN KEY(MACHINE_ID ) REFERENCES
                                              EQUIPMENT (MACHINE_ID )
ON UPDATE CASCADE;
```

ON UPDATE CASCADE

Before Update

```
SELECT * FROM LEASE WHERE MACHINE_ID = 'MAC01'
```

MACHINE_ID	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC01	INV001	7	20100.00	2019-04-01	2022-04-01
MAC01	INV002	3	35000.00	2020-02-01	2021-02-01

```
/* Update */
```

Update EQUIPMENT
SET MACHINE_ID = **'MAC09'** where MACHINE_ID = 'MAC01'

MACHINE_ID	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC99	INV001	7	20100.00	2019-04-01	2022-04-01
MAC99	INV002	3	35000.00	2020-02-01	2021-02-01

ON DELETE CASCADE

Before CASCADE

Delete from EQUIPMENT where MACHINE_ID = 'MAC99'


```
/* Drop the identified Lease foreign key in the above Error */
 ALTER TABLE LEASE DROP       FOREIGN KEY  lease_ibfk_1 ;
/* Add foreign key again */
 ALTER TABLE LEASE ADD
                        FOREIGN KEY(MACHINE_ID ) REFERENCES
                                              EQUIPMENT (MACHINE_ID )
ON UPDATE CASCADE
ON DELETE CASCADE ;
```

ON UPDATE CASCADE

Before Update

SELECT * FROM EQUIPMENT WHERE **MACHINE_ID = 'MAC02'**

MACHINE_ID	MACHINE_NAME	MODEL
MAC02	DYEING	2019

SELECT * FROM LEASE WHERE **MACHINE_ID = 'MAC02'**

MACHINE_ID	INVOICE	QUANTITY	UNIT_PRICE	LEASE_DATE	LEASE_EXPIRY
MAC02	INV003	4	15000.00	2019-06-01	2021-06-01

```
/* Delete */
DELETE FROM EQUIPMENT     WHERE MACHINE_ID = 'MAC02'

SELECT count(*) FROM LEASE  WHERE MACHINE_ID = 'MAC02'
```

count(*)
0

Referential cycles

Cycle of Referential Integrity constraints

- In a parent and Child table relationship,
  the primary **key of child table** references the primary key of the parent table.
- At same time, **primary key of parent table** references the primary key of the child table.
- So they form a loop of referential integrity constraints across the parent & child tables using their respective keys.

Example

Select * from ACCT_LOCKER

Acct_Num	Safe_box
4000-1956-2001	999101
4000-1956-2900	999102

Acct_Num: primary Key

Select * from BANK_LOCKER

BANK	Safe_box	Location	Acct_Num
HDFC	999101	HYD	4000-1956-2001
HDFC	999102	DELHI	4000-1956-2900

Safe_box : primary Key

Deferred Constraints

Deferred Constraints

- During large volume of transactions involving multiple dependencies, it is often difficult to process data efficiently due to the restrictions imposed by the constraints
- In order to override such constraints, to some extent DEFERRED constraints came into existence

Example

- Update of a primary key (PK) which is referenced by foreign keys (FK)
- In a primary key - foreign key relationship tables, parent table cannot update its primary key as it violates the referential integrity
- Similarly, child table cannot update its foreign key which may violates its parent table.
- The status is ***NOT DEFERRED*** and is default by SQL engine

Example

- Overriding the NON DEFERRED constraints with DEFERRED constraints
- These DEFERRABLE keyword can be defined in two ways
- One INITIALLY IMMEDIATE and other as INITIALLY DEFERRED
- INITIALLY IMMEDIATE updates the records directly onto database
- INITIALLY DEFERRED do not updates the records directly onto database rather it keeps the records in Logs . So that they can commit onto database later

Example

```
ALTER TABLE ACCOUNT_TRANSACTIONS
ADD constraint ACCOUNT_tran_init_imm
    FOREIGN KEY (Acct_Num) REFERENCES
ACCOUNT(Account_NUMBER) DEFERRABLE INITIALLY
IMMEDIATE;

ALTER SESSION SET CONSTRAINTS = DEFERRED;
```

Trying to update Foreign key

- Update `ACCOUNT_balance_det`

  SET Acct_Num = '9999-9999-9999' where account_number = '4000-1956-3456'

- Defer works are primarily used for Cyclic Foreign Keys under a scenario like

- Inserting a record in one table expects the record present in other table .

Trying to update Foreign key

- Usually the below insert will not work as they are referenced each other with primary - foreign keys
- Alter session as Deferred to allow insertions
- ALTER SESSION SET CONSTRAINTS = DEFERRED;
- Insert into ACCOUNT_Locker values ('4000-1956-2900' , 999102) ;
- Insert into Bank_Locker values (999102 , 'DELHI', '4000-1956-2900') ;

Thank You