

## 【初心者向け復習レポート】宿泊予約システムに見るデータベース設計とデータ保全の基礎（令和2年度秋期 応用情報技術者試験 問6）

### 1. システム要件と問題背景の直感的理解

みなさん、こんにちは！IT資格・データベース学習コンサルタントです。データベースの学習を進める中で、「文法は分かったけれど、実際のシステムでどう使われているのかイメージが湧かない」と悩んだことはありませんか？旅館や民宿などの宿泊予約を行うWebシステム（予約システム）において、ユーザーがストレスなく快適に部屋を予約できるようにするためには、現実世界の業務上のルール（ビジネスルール）を正確に理解し、それをシステムやデータベース構造へと落とし込むことが不可欠です。システム要件の定義があいまいであったり、データベース設計への反映が不十分であったりすると、二重予約（ダブルブッキング）やデータの矛盾といった、ビジネス上の致命的な不具合を引き起こしてしまいます。本問題で扱われる宿泊予約システムにおける利用者の操作手順およびシステムの振る舞い（図1「予約システムの主な要件」）は、以下のステップで整理されます。

- **ステップ1：利用者登録**
  - 利用者が予約システムを最初に利用する際、氏名、住所、電話番号を入力して「利用者登録」を行います。
- **ステップ2：空き部屋照会**
  - 利用者は条件入力画面で、希望する施設、チェックイン・チェックアウトの日付、予約したい部屋の種別、および部屋数を指定して空き状況を照会します。
- **ステップ3：空き状況の判定と画面遷移**
  - **空きがない場合：** 部屋が空いていない旨を画面に表示し、条件入力画面に戻って条件変更を促します。
  - **空きがある場合：** 予約手続画面に遷移します。利用者は宿泊人数を入力し、部屋の予約を確定します。
- **ステップ4：予約確定と排他制御（割り込み中断）**
  - 予約を確定するまでの間に他の利用者が先に予約を入れてしまい、必要な部屋数を確保できなくなった場合は、その旨を画面に表示して予約処理を中断します。

### ビジネスルール上の意味合い（So What?）

- **なぜ部屋種別ごとに予約を分けるのか？** システム要件として「予約は部屋の種別ごとに行う。種別の違う部屋を予約したい場合は、部屋の種別ごとに分けて予約を行う」と定められています。和室・洋室・スイートといった部屋種別ごとに管理単位を独立させることで、部屋の在庫確認や料金計算のロジックをシンプルかつ明確化するためです。
- **なぜ同時予約時に処理を中断するのか？** 照会時点では空室であっても、手続中に別のユーザーが同じ部屋を確保してしまうケース（競合状態）が発生します。この場合に無理に処理を進めると「部屋がないのに予約が完了する」という致命的な不整合が起きるため、システム側で明示的に処理を中断し、ユーザーに通知する安全設計を行っています。

## 要件とシステム振る舞いの対応関係

利用者のアクション, システムの振る舞い・処理

初回利用時にプロフ入力, 氏名・住所・電話番号をシステムに登録する

条件（施設・日付・種別・部屋数）を入力して照会, 指定された条件に合致する空き部屋の存在を確認する

（空き部屋なし）, 部屋が空いていない旨を表示し、条件変更を促す

（空き部屋あり）, 予約手続画面を表示し、宿泊人数の入力を求める

人数を入力して予約確定, 最終的な空室確認を行い、確保できなければ処理を中断。確保できれば予約を確定する

システム要件の基本を理解したところで、これらがどのようにデータベースの「形（構造）」に変換されるのかを次章で見ていきましょう。

## 2. E-R図の読み解きとデータベース構造の基礎

データベース設計の根幹をなすのが「E-R図（Entity-Relationship Diagram）」です。一見複雑な図に見えるかもしれませんが、コツさえ掴めば怖くありません！E-R図は、システム内で扱う現実世界のデータ（エンティティ：実体）と、それら相互の関係性（リレーションシップ）を視覚的に表現した設計図です。エンティティ同士の接続関係（1対1、1対多、多対多）や主キー・外部キーの設定を正しく把握することは、データの重複を防ぎ、矛盾のない整合したデータベースを構築するために極めて重要です。図2（データベースのE-R図）に示される各エンティティの役割と構造は以下の通りです。

| エンティティ名 | 主な属性（主キーはPK、外部キーはFK）                     | 役割と概要                            |
|---------|------------------------------------------|----------------------------------|
| 施設      | 施設ID（PK），施設名                             | 宿泊施設（旅館や民宿等）の基本情報を管理する           |
| 部屋      | 部屋ID（PK），施設ID（FK），部屋種別ID（FK），部屋番号        | 施設に存在する個々の部屋情報を管理する              |
| 部屋種別マスタ | 部屋種別ID（PK），名称，宿泊可能人数                     | 和室・洋室などの部屋のタイプや収容人数を管理する         |
| 利用者     | 利用者ID（PK），氏名，住所，電話番号                     | 予約システムを利用するユーザー情報を管理する           |
| 予約      | 予約ID（PK），利用者ID（FK），人数，チェックイン日付，チェックアウト日付 | 誰がいつ何人で予約したかという「予約の枠組み」を管理する     |
| 予約明細    | 予約明細ID（PK），予約ID（FK），部屋ID（FK），宿泊日，宿泊料     | 1回の予約に含まれる具体的な「部屋ごとの日別宿泊情報」を管理する |

設問1の解説（空欄a, b）

空欄a：予約 と 予約明細 のリレーションシップ

- 解答： → （「1対多」のリレーションシップ）
- 論理的解説： 利用者が1回の「予約」を行うと、滞在期間（チェックインからチェックアウトまで）や確保した部屋数に応じて、複数の「予約明細」レコードが作成

されます（例：2泊2部屋なら  $2 \times 2 = 4$  件の明細）。一方、作成された各「予約明細」レコードから見ると、それが所属する「予約」レコードは必ず1つに特定されます。したがって、予約 と 予約明細 の関係は「1対多」となり、予約 から 予約明細 に向かう矢印 →（1対多を示す記号）が入ります。

空欄b：部屋 エンティティに必要な属性

- 解答： 施設ID（外部キー）
- 論理的解説： 図2の凡例ルールにおいて、「実線下線は主キー（PK）」「破線下線は外部キー（FK）」と定義されています。施設 エンティティと 部屋 エンティティの間には「1対多」のリレーションシップが存在するため、部屋 テーブル側に親テーブルである 施設 の主キー（施設ID）を外部キーとして配置する必要があります。問題文中に「部屋IDは、全施設を通して一意な値である」と明記されているため、部屋 テーブルの主キー（PK）は 部屋ID（実線下線）のみで成立します。したがって、追加すべき 施設ID は主キーの一部ではなく、施設 テーブルを参照するための外部キーとして 破線下線（施設ID）で定義されます。

主キー（PK）と外部キー（FK）の役割（So What?）

- 主キー（Primary Key: PK）： テーブル内の行（レコード）を唯一無二（一意）に特定するための列です。データの重複や識別不能を防ぎます。
- 外部キー（Foreign Key: FK）： 別のテーブルの主キーを参照する列です。テーブル間のつながりを保証し、存在しない施設IDや予約IDが登録されるのを防ぐ「参照整合性」を保ちます。これらを適切に配置することで、データ構造の破綻を防ぎ、複雑な宿泊データの一貫性を保つことができます。テーブル構造が定義されたところで、次に実際に空き部屋を検索するSQL文のロジック解説へ進みましょう。

### 3. 空き状況確認SQLの解説（NOT EXISTSとHAVING句）

SQLのコードを見ると「うっ、難しそう…」と感じてしまうかもしれませんが、大丈夫です！どんなに長く見えるSQL文でも、句（WHERE、GROUP BY、HAVINGなど）ごとにパーツを分解して紐解けば、パズルのように整然とした論理構造が見えてきます。図3に示す「部屋の空き状況の確認を行うためのSQL文」は、指定された条件（施設、部屋種別、期間、必要部屋数）を満たす空室が存在するかどうかを判定する心臓部です。

```
SELECT 施設ID, 部屋種別ID, COUNT(*) FROM 部屋
WHERE NOT EXISTS ( -- 空欄c
    SELECT * FROM 予約明細
    WHERE 予約明細.部屋ID = 部屋.部屋ID
        AND 予約明細.宿泊日 >= :チェックイン日付
        AND 予約明細.宿泊日 < :チェックアウト日付
)
AND 施設ID = :施設ID
AND 部屋種別ID = :部屋種別ID
GROUP BY 施設ID, 部屋種別ID
```

HAVING COUNT(\*) >= :部屋数 -- 空欄d

ドメイン補足：なぜ 宿泊日 < :チェックアウト日付（未満）なのか？

実務ドメインの知識として、ホテルの「チェックアウト日」には宿泊しません。たとえば「8月1日チェックイン～8月3日チェックアウト」の2泊の場合、実際の宿泊日は「8月1日」と「8月2日」の2日間です。そのため、SQLの条件式は <= :チェックアウト日付（以下）ではなく、< :チェックアウト日付（未満：半开区間）という境界条件になります。丸暗記ではなく、実際のホテルの運用と結びつけると納得しやすいですね！

設問2の解説（空欄c, d）

空欄c : NOT EXISTS による除外ロジック

- **解答：** NOT EXISTS
- **解説：** 副問合せ（サブクエリ）では、予約明細 テーブルを参照し、「対象の部屋（部屋. 部屋ID）において、指定された宿泊期間内に既に予約が入っている明細レコード」を検索しています。主問合せ（アウタークエリ）側で求めたいのは「空いている部屋」です。したがって、副問合せでヒットした（＝すでに予約が存在する）部屋を **除外** しなければなりません。サブクエリの結果が存在しないことを判定する演算子である NOT EXISTS を指定することで、「指定期間中に予約明細が存在しない部屋」のみを抽出できます。

【初心者向け補足】 EXISTS と NOT EXISTS の違い

- EXISTS: サブクエリの条件に **一致するレコードが存在する場合** に条件を満たす (True)。
- NOT EXISTS: サブクエリの条件に **一致するレコードが存在しない（空である）場合** に条件を満たす (True)。

空欄d : HAVING COUNT(\*) による集計後の条件判定

- **解答：** HAVING COUNT(\*)
- **解説：** SQLでは、GROUP BY 施設ID, 部屋種別ID によって施設および部屋種別ごとに空き部屋がグループ化されています。グループ化された後の集計値（ここでは空室の合計数 COUNT()）に対して「希望する部屋数（:部屋数）以上あるか」という条件判定を行うためには、WHERE 句ではなく HAVING 句を使用する必要があります。したがって、HAVING COUNT() >= :部屋数 と記述することで、希望部屋数以上の空室が確保できる場合のみレコードを返すロジックが完成します。

検索ロジックの4ステップ

1. **部屋の絞り込み（WHERE句）：** 指定された 施設ID かつ 部屋種別ID である部屋を対象とする。
2. **予約重複の除外（NOT EXISTS）：** チェックイン日からチェックアウト日直前までの期間に、すでに 予約明細 が存在する部屋を取り除く（＝空室のみを残す）。

3. **グループ化 (GROUP BY) :** 残った空室を 施設ID と 部屋種別ID でグループにまとめる。
4. **必要数の判定 (HAVING COUNT(\*)) :** グループごとの空室数 COUNT(\*) が、ユーザーの希望する :部屋数 以上存在するかをチェックする。空き状況の確認ロジックを理解したところで、次にシステム運用上で発生した「同時実行による不具合」とその解決策について考察してみましょう。

#### 4. 同時実行不具合 (レースコンディション) と一意性制約による防止

Webシステムのように、世界中から複数のユーザーが同時にアクセスして同じデータベースを操作する環境では、「マルチスレッド／同時実行によるデータ不整合 (レースコンディション)」への配慮が不可欠です。

テスト時に発生した不具合の原因

システムテストにおいて、「同じ宿泊日の同じ部屋について、予約明細のレコードが重複して挿入されてしまう (ダブルブッキング)」という恐ろしい不具合が報告されました。原因は、**別々の利用者が「全く同じ時刻」に予約確定の操作を行ったこと**にあります。予約確定の処理 (図5) では、挿入前に図3のSQLを実行して空室を確認した上でレコードを挿入します。しかし、全く同じタイミングで処理が走ると、以下のようなすり抜けが発生します。

1. ユーザーAとユーザーBが同時に予約確定ボタンを押す。
2. ユーザーAの処理で空き状況確認SQLが走り、「空きあり」と判定される。
3. ほぼ同時にユーザーBの処理でも空き状況確認SQLが走り、ユーザーAの挿入が完了 (コミット) する前であるため「空きあり」と判定されてしまう。
4. ユーザーA・ユーザーBの両方の処理が「予約可能」と信じてそのまま進み、同じ部屋・同じ宿泊日に対して重複して予約明細レコードが挿入されてしまう。

設問3(1), (2)の解説 (空欄e, f, g)

このようなタイミングの差による重複登録を防止するため、データベースの機能を用いて強制的に重複を拒否する設定を行います。

空欄e, f: 複合キーを構成する列名

- **解答 :** 宿泊日, 部屋ID (順不同)
- **解説 :** 「同じ宿泊日の同じ部屋」に二重で予約が入ることがビジネス上の根本問題です。したがって、予約明細 テーブルにおける 宿泊日 列と 部屋ID 列の2つの組み合わせ (複合キー) に対して重複を許さない設定を行う必要があります。

空欄g: 制約の名称

- **解答 :** UNIQUE制約 (または 一意性制約)
- **解説 :** 指定した列 (または列の組み合わせ) の中で、値の重複を許さないようにデータベースレベルで強制する機能を UNIQUE制約 (一意性制約) と呼びます。

DB側で「一意性制約」をかける重要性 (So What?)

プログラム (アプリケーション) 側のチェックロジックだけに頼った排他制御は、ミリ秒単位の極めて微妙なタイミングのズレによってチェックをすり抜けてしまう危険性が常に残ります。データベース自体に UNIQUE制約 を設定しておけば、万が一アプリ側のチェックを同時にすり抜けて挿入が試みられたとしても、データベースエンジンがエラーを検知して重複レコードの挿入を物理的にブロックします。これこそが、システムの堅牢性 (データの整合性) を守る強力な砦となります。不具合を防ぐための恒久対応 (UNIQUE制約の追加) を行う前に、すでに発生してしまった「重複データの削除 (クレンジング)」が必要になります。そのSQLロジックを最終章で解説します。

## 5. 重複データのクレンジング処理 (相関副問合せとMIN関数)

「相関副問合せ」と聞くと身構えてしまう方も多いかもしれませんが、実際のデータを使ったシミュレーションを行ってみれば驚くほどシンプルです！データベースに既に重複した不整合データが存在している状態では、新たに UNIQUE制約 を追加しようとしてもデータベースから「すでに重複があるため制約をつけられません」とエラーを返されてしまいます。そのため、制約を追加する前に、既存の不整合データを正しく特定して削除する「データクレンジング」が必要となります。図6に、削除対象となる重複レコードを抽出するSQL文が示されています。

```
SELECT t1.予約ID, t1.予約明細ID, t1.部屋ID, t1.宿泊日 FROM 予約明細 t1
WHERE t1.予約ID > (
    SELECT MIN(t2.予約ID) -- 空欄h
    FROM 予約明細 t2
    WHERE t1.部屋ID = t2.部屋ID -- 空欄i
    AND t1.宿泊日 = t2.宿泊日 -- 空欄j
)
```

### 削除方針の確認

問題文の決定事項として、「同じ宿泊日の同じ部屋の予約が重複した予約明細のレコードについて、最初に挿入された予約のレコード (および紐づく予約明細) を残し、それ以外の予約明細、予約のレコードを削除する」と決められています。

設問3(3)の解説 (空欄h, i, j)

空欄h : 最古の予約IDを求める集計関数

- **解答 :** MIN(t2. 予約ID)
- **解説 :** 前提条件として「予約ID, 予約明細IDは, レコードを挿入した順に値が大きくなる」と示されています。つまり、「最初に挿入された予約」とは、重複しているレコード群の中で **最小の予約ID** を持つレコードを意味します。したがって、最小値を求める集計関数 MIN を用いて MIN(t2. 予約ID) と指定します。

空欄*i, j* : 相関副問合せの結合条件

- **解答：** t1. 部屋ID = t2. 部屋ID , t1. 宿泊日 = t2. 宿泊日 (順不同)
- **解説：** 主問合せ (外側の t1) で確認しようとしているレコードに対して、副問合せ (内側の t2) で「同じ部屋かつ同じ宿泊日」を持つレコード群を比較対象として絞り込む必要があります。このように主問合せの列と副問合せの列を結びつけて処理する手法を「相関副問合せ」と呼びます。条件式として t1. 部屋ID = t2. 部屋ID かつ t1. 宿泊日 = t2. 宿泊日 を指定することで、同一の部屋・宿泊日グループにおける「最小の予約ID」が求められます。

直感的に分かる！具体例 (数値シミュレーション)

具体的なデータ例で判定ロジックを追ってみましょう。例えば、\*\*「部屋101・8月1日」\*\*の宿泊に対し、同時の予約確定によって **予約ID: 101** と **予約ID: 105** の2つの予約明細が重複して作成されてしまったとします。

【予約明細テーブルのデータ例】

- ・ 予約明細ID: 501 / 予約ID: 101 / 部屋ID: 101 / 宿泊日: 8月1日 (先に作成)
- ・ 予約明細ID: 505 / 予約ID: 105 / 部屋ID: 101 / 宿泊日: 8月1日 (後から重複作成)

SQL実行時、主問合せ (t1) の1行ごとに以下のチェックが行われます。

- ① t1 が「予約ID: 101」のレコードを評価する場合：
  - 副問合せで「部屋101・8月1日」の最小予約IDを検索  $\rightarrow$  MIN(t2. 予約ID) は 101 と算出される。
  - 条件式 t1. 予約ID > 101 (つまり 101 > 101) を評価  $\rightarrow$  偽 (False)。
  - 判定：抽出されない (=削除されずに正しく残る！)。
- ② t1 が「予約ID: 105」のレコードを評価する場合：
  - 副問合せで「部屋101・8月1日」の最小予約IDを検索  $\rightarrow$  MIN(t2. 予約ID) は 101 と算出される。
  - 条件式 t1. 予約ID > 101 (つまり 105 > 101) を評価  $\rightarrow$  真 (True)。
  - 判定：抽出される (=後から重複して入った不要なレコードとして削除対象になる！)。このように、最小の予約ID (一番古い予約) 以外の重複レコードだけが見事に抽出され、削除対象となるロジックです。このSQLによって抽出された不要な予約明細レコード、およびそれに紐づく予約レコードを削除することで、データベースが綺麗な状態に戻り、無事に UNIQUE制約 を適用できるようになります。

## 6. まとめ：応用情報技術者試験におけるDB問題の攻略ポイント

お疲れ様でした！本レポートでは、令和2年度秋期応用情報技術者試験の間6を題材に、宿泊予約システムの要件定義からE-R図の解説、空室確認SQL、さらには同時実行不具合 (レースコンディション) への対応とデータクレンジングまでをステップバイステップで網羅しました。IPAの出題趣旨 (SOURCE\_IMAGE\_9) を分析すると、試験で問われている本質は大きく次の\*\*「2つの軸」\*\*に集約されます。

本問題を通じて得られる「2つの軸」の学び

① 基本理解軸 (E-R図・SQL解読)

- **実務要件の概念化:** 宿泊システムのビジネスルールを、E-R図の「1対多(→)」や「主キー／外部キー」の記号ルールへ正しくマッピングできること。
- **検索ロジックの解読:** NOT EXISTS による不適合データの除外や、GROUP BY・HAVING 句による集計後条件指定など、標準SQL文のデータ抽出フローを正確に追えること。

② データ保全・応用力軸 (排他制御・不具合対応・クレンジング)

- **同時実行課題の洞察:** Webシステム特有のミリ秒単位の競合状態(レースコンディション)によってアプリ側のチェックがすり抜ける危険性を察知できること。
- **恒久対応とデータ整備:** 重複を防ぐDBレベルの UNIQUE制約(一意性制約)の重要性を理解し、制約適用前に相関副問合せと集計関数(MIN)を用いて不整合データを安全に削除(クレンジング)できること。

解答一覧(設問1～設問3)

設問, 空欄, 公式解答例, 備考

設問1, a, →, 1対多のリレーションシップ

, b, 施設ID, 外部キー(破線下線)

設問2, c, NOT EXISTS, 予約が存在しない部屋を抽出

, d, HAVING COUNT(\*), 集計後の部屋数条件判定

設問3 (1), e, 宿泊日, 順不同(fと入れ替え可)

, f, 部屋ID, 順不同(eと入れ替え可)

設問3 (2), g, UNIQUE制約(または一意性制約), DBレベルの重複防止制約

設問3 (3), h, MIN(t2. 予約ID), 最初に挿入された最古のID

, i, t1. 部屋ID = t2. 部屋ID, 順不同(jと入れ替え可)

, j, t1. 宿泊日 = t2. 宿泊日, 順不同(iと入れ替え可)

初学者に向けた: 試験突破のための3つの心得

- **1. ビジネスルールとE-R図の記号(矢印・下線)を厳密に対比させる** 「1対多(→)」の関係や「主キー(実線下線)」「外部キー(破線下線)」の凡例ルールをしっかりと意識しましょう。問題文中の前提条件(「部屋IDは全施設を通して一意」など)に必ずヒントが隠されています。
- **2. SQL文は「どのテーブルから」「どんな条件で絞り込み」「どう集計しているか」を分解して読む** NOT EXISTS による不適合データの除外や、GROUP BY と HAVING 句の使い分けなど、各述語や句の役割を頭の中でシミュレーションしながら紐解きましょう。
- **3. 同時実行のトラブル(レースコンディション)と制約の役割をセットで押さえる** 「アプリ側のチェックをすり抜けるタイミング問題」を意識し、データベースの UNIQUE制約(一意性制約)がデータ整合性を守る「最後の砦」となるというシステム設計思想を理解しておきましょう。データベースの概念は、一度仕組みが分かると

実務でも試験でも強力な武器になります。焦らず1つずつ構造を紐解いて、確実な実力を身につけていきましょう！応援しています！